TECHNICAL INFORMATION

TECHNICAL INFORMATION

Introduction

This section explains TRSDOS-II on a technical level. It teaches you how to use TRSDOS-II system routines within your own Z-8Ø assembly language programs. You also may find the information useful in programming with BASIC or other languages.

Memory Requirements

TRSDOS-II resides on your primary drive; it occupies only a small portion of memory at any one time. The supervisor program, input/output drivers, and other essentials are always in memory. Auxiliary code is loaded as needed into an "overlay area."

Memory addresses Ø-1Ø239 (X'ØØØØ'-X'27FF') are reserved for TRSDOS-II.

All TRSDOS-II utilities use memory in the range of X'28ØØ' -X'2FFF'.

The following utilities use all of user memory: FORMAT, BACKUP, FCOPY {SYS} and single-drive COPY.

You must locate user programs above X'27FF'. You may want to locate them above X'2FFF', so you can use the utility overlays without losing your program.

Memory Requirements of TRSDOS-II

Decimal Address	Area	Hex Address
Ø-1Ø239	TRSDOS-II Resident Area	ØØØØ-27FF
1Ø24Ø-12287	Utility Overlay Area	28ØØ-2FFF

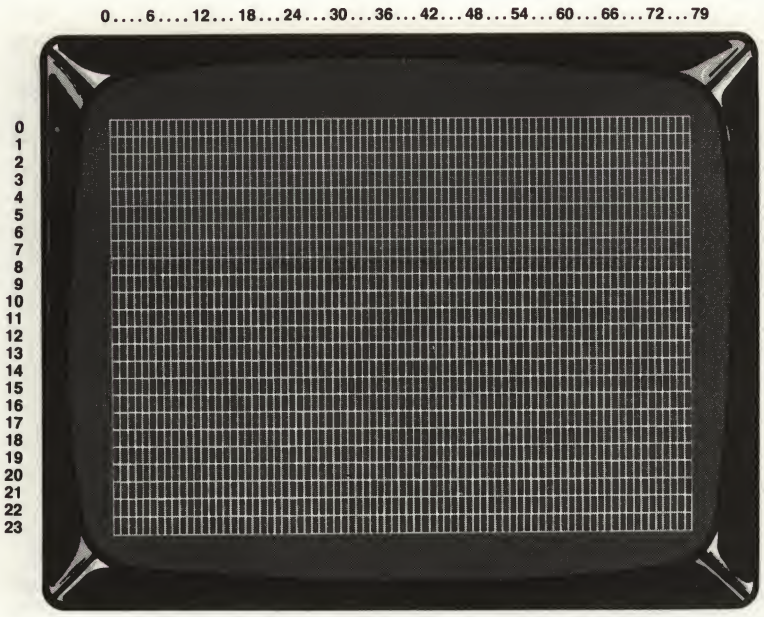
Decimal Address	Area	Hex Address
12288-61439	User Area	3ØØØ-EFFF
6144Ø-65535	TRSDOS-II Demand Resident Area	FØØØ-FFFF

Video Display

The display has two modes of operation -- scroll and graphics. Cursor motion and allowable input characters are different in the two modes.

Graphics Mode

In the graphics mode, the display can be thought of as an 8Ø by 24 matrix, as illustrated below:



DISPLAY POSITIONS, GRAPHICS MODE

Note: The display has two character sizes: 8Ø characters per line and 4Ø characters per line. The above illustration shows the 8Ø characters-per-line mode.

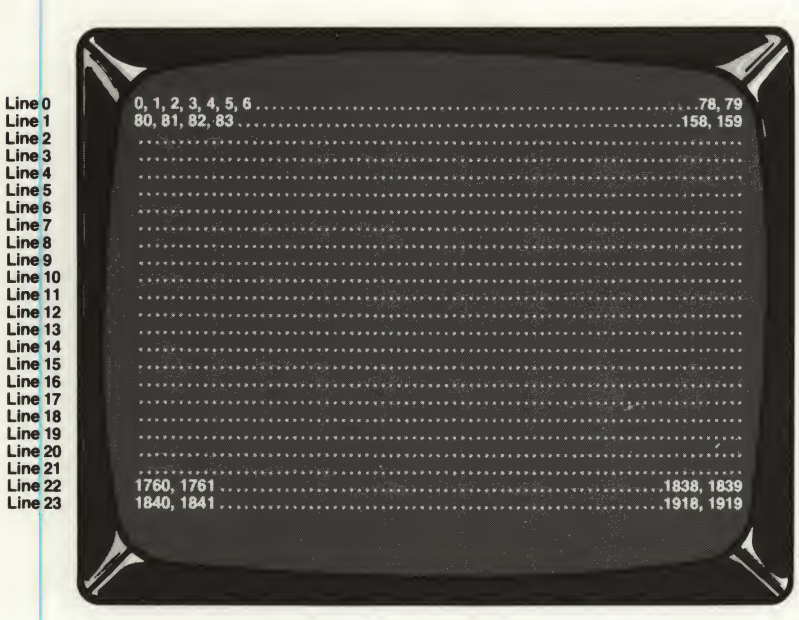
Each time an acceptable character is received, it is displayed at the current cursor position, which is set upon entry to the graphics-mode routines. After the character is displayed, the cursor position is advanced, as follows:

> If the cursor is to the left of Column 79, it advances to the next column position on the same row.
>
> If the cursor is at Column 79, it wraps around to Column Ø on the next row.

Cursor movement works the same way in all directions. For example, if the cursor is at Row 23, Column 4Ø, and TRSDOS-II receives the X'FF' (graphics-down) code, the cursor wraps around to Row Ø in the same column.

Scroll Mode

In the scroll mode, the display can be thought of as a sequence of 192Ø display positions, as illustrated below:



DISPLAY POSITIONS, SCROLL MODE

Note: The display has two character sizes: 8Ø characters per line and 4Ø characters per line. The illustration above shows the 8Ø characters-per-line mode.

Each time an acceptable character is received, it is displayed at the current cursor position. Then, the cursor advances to the next higher numbered position.

When the cursor is on the bottom line, and a line feed or carriage return is received, or the bottom line is filled, the entire display is "scrolled." TRSDOS-II:

1. Deletes Line Ø

2. Moves Lines 1-23 up one line

3. Blanks Line 23

4. Sets the cursor at the beginning of Line 23

Note: You can use the SCROLL SVC to protect Lines Ø to 22 from scrolling.

Diskette Organization

TRSDOS-II can use either single- or double-sided, double-density diskettes. It automatically formats each diskette according to its type.

Each side of the double-sided diskette contains 77 tracks, numbered Ø-76. Therefore, you can think of the double-sided diskette as one diskette with 154 tracks.

The single-sided diskette has 77 tracks on one side only.

The sector is the basic unit of space allocation in TRSDOS-II. Except for Track Ø, each track contains 32 sectors, numbered 1-32. Each sector contains 256 bytes.

TRSDOS-II reserves Track Ø for itself and formats it single-density. Track Ø contains 26 sectors and 128 bytes per sector. On double-sided diskettes, only one side has a single-density Track Ø.

The capacity of a double-sided diskette is:

(153 * 32 * 256) + (1 * 26 * 128) = 1,256,7Ø4 bytes

The capacity of a single-sided diskette is:

(76 * 32 * 256) + (1 * 26 * 128) = 625,92Ø bytes

Single-sided Diskette	Tracks	Sectors	Bytes
1	76	2,432	622,592
---	1	32	8,192
---		1	256

Double-sided Diskette	Cylinders	Tracks	Sectors	Bytes
1	77	153	4,896	1,253,376
---	1	2	64	16,384
---	---	1	32	8,192
---	---	---	1	256

Disk Files

Methods of File Allocation

TRSDOS-II gives you two ways to allocate disk space for files: dynamic allocation and pre-allocation.

Dynamic Allocation

With dynamic allocation, TRSDOS-II allocates sectors only when the file is written to. For example, the first time you open a file for output, no space is allocated. The first space allocation is done at the first write. TRSDOS-II allocates more space as required by later writes.

With dynamically allocated files, TRSDOS-II de-allocates (recovers) unused sectors when you close the file.

Pre-Allocation

With pre-allocation, you specify the number of sectors to be allocated. The only way to do this is by using the CREATE command when you create the file.

When the file becomes too large for the space allocated to it, TRSDOS-II dynamically extends (enlarges) it as needed.

It does not, however, de-allocate unused sectors when you close a pre-allocated file. To reduce the size of a pre-allocated file you must copy it to a dynamically allocated file. The COPY command does this automatically when the destination is a dynamically allocated file.

Record Length

TRSDOS-II transfers data to or from a diskette one sector (256 bytes) at a time. These data blocks are called "physical records."

TRSDOS-II transfers data to or from a file in buffers which are from 1 to 256 bytes long. These buffers are called "user records" or "logical records."

TRSDOS-II automatically "blocks" your logical records into physical records to be transferred to the disk. It also automatically "de-blocks" the physical records into logical records, which are used by your program.

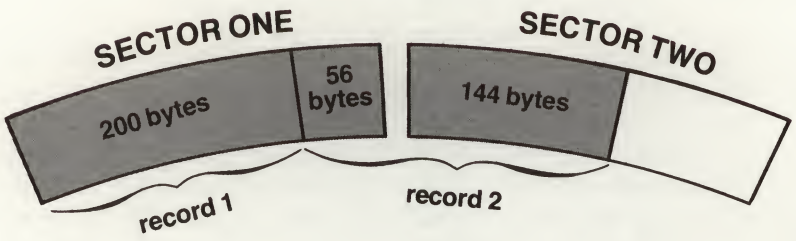
Therefore, your only concern during file access is with logical records. You never need to worry about physical records, sectors, tracks, and so on. This is to your benefit, since physical record lengths and features may change in later TRSDOS-II versions, while the concept of logical records will not.

From this point on, the term "record" refers to a "logical record."

Spanning

If the record length is not an even divisor of 256, the records automatically are spanned across sectors.

For example, if the record length is 2ØØ, Sectors 1 and 2 look like this:



Fixed-Length and Variable-Length Records

TRSDOS-II files can have either fixed-length or variable-length records. Files with fixed-length records are called FLR files; files with variable-length records, VLR files.

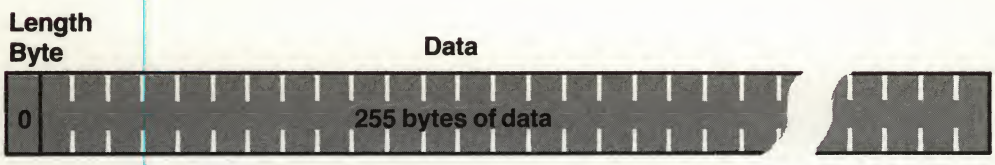
For an FLR file, you set the record length when you open the file for the first time. This length can be from 1-256 bytes. You cannot change the set record length unless you overwrite the file with new data.

For a VLR file, you specify the record length in a 1-byte length field at the start of each record. The lengths of records in a VLR file can vary. For example, the first record in a file might have a length of 32; the second record, 17; the third record, 25Ø; and so on.

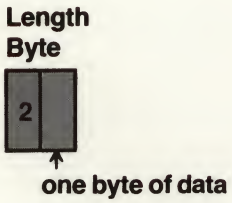
The length byte for a variable-length record indicates the entire length of the record, including the length byte. It can be from Ø-255. You can use 1, but it has no meaning because no record is 1-byte long.

Examples

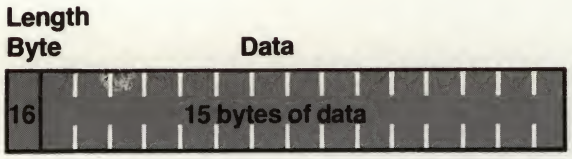
A length byte of Ø indicates that the record has 255 bytes of data:



A length byte of 2 indicates that the record has 1 byte of data:



A length byte of 16 indicates that the record has 15 bytes of data:



Record Numbers

Records are numbered from Ø (beginning of file) to 16,777,214.

A file may contain up to 16,777,216 bytes of storage. Therefore, to determine the number of records a file can hold, use the following formula:

16,777,216 / logical record length = number of records

For example:

16,777,216 / 38 = 441,5Ø5 records.

For a file with a record length of 1, the number of records cannot exceed 16,777,214.

The record number 16777215 (X'FFFFFF') positions to the "end of file" (EOF), regardless of the actual number of records in the file.

Record Processing Capabilities

TRSDOS-II allows both direct and sequential file access.

Direct access, sometimes called "random access," lets you process records in any order you specify.

To get direct access to Records Ø-65533, you simply specify the record number (when using one of the direct access SVCs).

To get direct access to Records 65534-16777216, however, you must choose the extended file access mode when you open the file. This mode specifies the record number in three bytes, letting you use the large number.

The direct access SVCs are DIRRD (direct-read) and DIRWR (direct-write).

Sequential access lets you process records in the order in which they are stored in the file. TRSDOS-II automatically accesses the record that follows the last one processed. When you first open the file, it accesses Record Ø.

FLR files can be opened by either direct access or sequential access.

VLR files can be opened only by sequential access. You cannot position to a specific record, since the varying record lengths make it impossible to calculate the record's position.

The sequential access SVCs are READNX (read-next) and WRITNX (write-next). When you first open the file, sequential processing starts with Record Ø. However, you can use DIRWR and DIRRD to position to the EOF.

Examples with FLRs

Assume you have an open FLR file. Here are some typical sequences you can do via the file processing routines:

- **Read and/or write records in the file -- in any order.**
  Using DIRRD and DIRWR, you can: read Record 5, write at the EOF, read Record 3, write Record 3, and so on.

- **Read or write sequentially, starting anywhere in the file.**
  Do a DIRRD to the record at which you want to start. Then, do READNX or WRITNX until done.

- **Write sequentially, starting at EOF.**
  Do a DIRWR to the EOF. Then, do WRITNX until done.

- **Learn the number of records in a file.**
  Do a DIRRD to the EOF. Then, use the LOCATE routine to get the current record number. This equals the number of records + 1.

Examples with VLRs

Assume you have an open VLR file. Here are some typical sequences you can do via file processing routines:

- **Read or write sequentially, starting at the first record.**
  Do READNX and WRITNX until done.

- **Write sequentially, starting at the EOF.**
  Do a DIRWR to the EOF. Then, do WRITNX until done.

Note: Whenever you write to a VLR, the last record you write becomes the end of file (EOF), automatically.

You cannot update a VLR file directly. You must read in the file and output the updated information to a new VLR file.

Supervisor Calls

Supervisor calls (SVCs) are operating system routines available to any user program. They alter certain system functions and conditions; let you access files; do input/output to the keyboard, video display, and printer; and do various computations.

The available TRSDOS-II SVCs are:

KEYBOARD SVCs
KBCHAR
KBINIT
KBLINE
KBPUT
VIDKEY

VIDEO SVCs	
CURSOR	VDINIT
SCROLL	VDLINE
VDCHAR	VDREAD
VDGRAF	VIDRAM

PRINTER SVCs
PCTRL
PRCHAR
PRINIT
PRLINE

COMMUNICATIONS SVCs	
ACTRL	BCTRL
ARCV	BRCV
ATX	BTX
	RS232C

SYSTEM CONTROL SVCs	
CLRXIT	HLDKEY
DATE	INITIO
DELAY	JP2DOS
DOSCMD	RETCMD
ERRMSG	SETBRK
ERROR	SETUSR
	TIMER

FILE ACCESS SVCs	
CLOSE	LOCATE
DIRRD	OPEN
DIRSET	RAMDIR
DIRWR	RDDIR
DISKID	READNX
FILPTR	RENAME
KILL	REWIND
	WRITNX

COMPUTATIONAL SVCs
BINDEC
BINHEX
MPYDIV

MISCELLANEOUS SVCs	
LOOKUP	SORT
PARSER	STCMP
RANDOM	STSCAN
SOUND	WILD

Note: Appendices C and D, the SVC quick reference lists, summarize each SVC's function.

Each SVC has a function code that you use to call it. These codes range from Ø through 127. The first 96 codes (Ø-95) are defined by TRSDOS-II. Codes 96-127 are available for you to define (see the SETUSR SVC).

SVC Notation

In this section we use the following notations:

Notation	Meaning
RP = data	Register Pair RP contains the data
nl < R < n2	The register pair contains a value greater than nl and less than n2
(RP) = data	Register Pair RP contains the address of ("points to") the data
NZ = Error	If the Z flag is not set, an error occurred

When a range is not given in the description of the SVC, you can use any representable number. For example, a register can contain any value from Ø-255.

Calling Procedure

All SVCs are executed via the **RST 8** instruction.

1. Load the function code of the desired SVC into Register A. Also load any other registers that the SVC needs. See "Supervisor Calls."

2. Execute an RST 8 instruction.

3. If the function is successful, the Z flag is set upon return from the SVC. If the Z flag is not set, there is an error. Register A has the appropriate error code (except after certain computational SVCs, which use Register A to return other information).

During the execution of an SVC, the SVC processor does not alter any memory above X'2FFF'. Only those Z-8Ø registers used to pass parameters from the SVC are changed. However, all the prime registers are used by TRSDOS-II; they are not restored.

Examples

Time-Delay

```
      LD    BC,TIMCNT            ; LENGTH OF DELAY
      LD    A,6                  ; FUNCTION CODE 6 = DELAY-SVC
      RST   8                    ; JUMP TO SVC
;   DELAY OVER-PROGRAM CONTINUES HERE
```

Output a Line to the Video Display

```
      LD    HL,MSG               ; POINT TO THE MESSAGE
      LD    B,1Ø                 ; B = CHARACTER COUNT
      LD    C,ØDH                ; C=CTRL CHAR. TO ADD AT END
      LD    A,9                  ; CODE 9 = DISPLAY LINE-SVC
      RST   8                    ; JUMP TO SVC
      JR    NZ, GOTERR           ; JUMP IF I/O ERROR
;   IF NO ERROR THEN PROGRAM CONTINUES HERE
```

Get a Character from the Keyboard

```
GETCHAR  LD     A,4              ; CODE 4 = GET CHARACTER-SVC
         RST    8                ; JUMP TO SVC
         JR     NZ,GETCHAR       ; DO AGAIN IF NO CHARACTER
;   CHARACTER IS IN REGISTER B
```

Using the Serial Communications SVCs

Initialization

Before doing any serial I/O, you must initialize the channel you want. Use either the SETCOM library command or the RS232C SVC.

Input/Output

All serial I/O is character-oriented, similar to keyboard input and video output. The main difference is that your program must check the communications status at various times. This is done to ensure that the communications link is in place and that data is not being lost.

After any attempted serial I/O, TRSDOS-II returns the communications status to your program in Z-8Ø Register A. The bits in the register are used individually to show the on/off status of various conditions. Certain bits apply to transmit operations; others apply to all serial I/O.

Programming with TRSDOS-II

This section tells you how to execute your own machine-language programs under TRSDOS-II. It has two sub-sections:

- **Program Entry Conditions** -- How control is transferred to your program after the program is loaded from the disk
- **Handling Programmed Interrupts** -- How to write an interrupt service routine for <BREAK> key processing and TIMER interrupts

To create and use a program, follow this procedure:

1. Enter the program into memory, using DEBUG, or via the serial interface channel from another device, or using an assembler and linker.
2. If you use DEBUG or an I/O channel to load the program, use the DUMP command to save the program as an executable disk file, setting the load and transfer addresses.
3. To run the program, input the filename to the TRSDOS-II command interpreter (TRSDOS-II Ready mode).

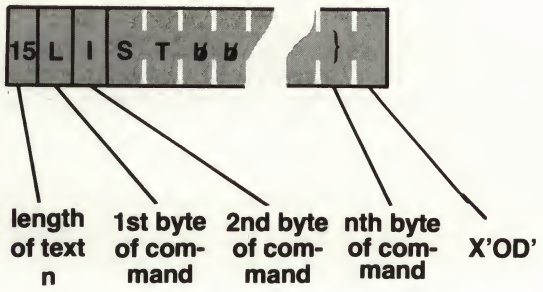
Program Entry Conditions

Upon entry to your program, TRSDOS-II sets up the following registers:

BC = address of the first byte following your program (the first free byte for use by your program).
DE = highest unprotected memory address (the end of memory that can be used by your program).

HL = address of the buffer containing the last command entered to the TRSDOS-II command interpreter.

The first byte of the buffer contains the command line length, excluding the carriage return. The text of the command follows this length byte. For example:



Handling Programmed Interrupts

TRSDOS-II allows two user-programmed interrupts as described under SETBRK and TIMER. When either kind of interrupt is received (you press the <BREAK> key or the TIMER counts to zero), control transfers to your interrupt handling routine.

Note: System routines called by your program or called by interrupt handlers can be interrupted, also.

Upon entry to your interrupt processing routine, TRSDOS-II sets up the registers as follows:

(SP) = address of the next instruction to be executed when the interrupt was received.

Other registers:
Contents are the same as they were when the interrupt was processed.

Before doing any processing, you should save all registers. When you finish processing, restore all registers and execute a return to continue with the interrupted program.

Keep the interrupt handling routines short. Ideally, the routine simply flags the main program that an interrupt has

occurred, and then returns. The main program then can respond to the interrupt flag when convenient.

Always end your interrupt handler with the RET instructions and with all registers intact.

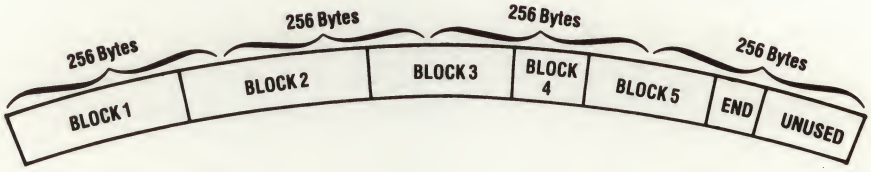
TRSDOS-II is serially reusable but not always re-entrant. Specifically, your interrupt routine should not call TRSDOS-II SVCs, since under some conditions this causes unpredictable results.

Program Files

This section describes the required format and structure of program files. In it you'll also find a summary of the procedure for writing a program file using TRSDOS-II file-access SVCs.

Program File Format

A program file is stored on the diskette in blocks, which might look like this:



The lengths of the blocks vary, depending on the type of block and the amount of information in it. The blocks must border each other. There can be no unused bytes between them.

There are three major types of blocks:

- **Program data blocks** -- These contain the actual program data, prefixed by four bytes of header information.
- **Comment blocks** -- These contain documentation for the programmer. Comment blocks are not loaded or examined by the loader. Comment blocks are prefixed by two bytes of header information.

- **Trailer blocks.** Each program file ends with a trailer block. It marks the end of file (EOF) and tells TRSDOS-II what to do after the file has been loaded -- either to transfer to a specified address or to return to the caller. Trailer blocks are four bytes long.

The first byte in a block identifies the block type, as follows:

Contents (Hex)	Block Type
Ø1	Program data
Ø5	Comment block
Ø2	Trailer block; jump after loading
Ø3	Trailer block; return to the caller after loading
ØØ, Ø4, Ø6 - FF	Reserved for TRSDOS-II use; the loader fails if one of these codes is used

Structure of Program Data Blocks

Program data blocks consist of the following:

Byte #	Contents
1	Block identifier (equals binary 1)
2	Length (the number of bytes of program data plus two for the load address)
3-4	Load address (where the following program data starts loading into RAM); it must be in the LSB-MSB format
5-end	Program data

The length byte gives the number of bytes in the rest of the block -- following the 2-byte load address. This sum may range from 3 (the 2-byte address plus 1 byte of data) to 258 (the 2-byte address plus 256 bytes of data). You must translate the sum into the range Ø-255.

To do this, take the number of program bytes and increment by two. Note that values greater than 255 "wrap around" to Ø, 1, and 2. Here is a table:

Number of Bytes after Address (program data only)	Value for Length Byte
1	3
2	4
3	5
...	...
253	255
254	Ø
255	1
256	2

The load address tells TRSDOS-II where in RAM to load the data in the block. TRSDOS-II loads the bytes serially. The load address must allow the entire block of program data to load in the "user" area of RAM.

Structure of Comment Blocks

Comment blocks consist of the following:

Byte #	Contents
1	Block identifier (equals binary 5)
2	Length (the number of bytes in the comment)
3-end	Comment

The length byte gives the number of bytes in the comment (the number of bytes after the length itself). This sum ranges from Ø-255.

Length Byte	Comment Length
Ø	Ø
1	1
2	2
...	...
255	255

Structure of Trailer Blocks

Each program file ends with a single trailer block that tells TRSDOS-II what to do next:

Jump to a specified "transfer" address
or
Return to the caller

Trailer blocks consist of the following:

Byte #	Contents
1	Block identifier (equals binary "2" or "3")
2	Length (equals 2)
3-4	Transfer address; it must be in the LSB-MSB sequence and must be in the user area of RAM

If the block identifier equals 2, TRSDOS-II jumps to the transfer address after loading the program. If it equals 3, TRSDOS-II does not jump to the transfer address. Instead, it returns to the caller after loading the program, with the following prime registers set:

HL' = transfer address taken from the trailer block
DE' = address of the last usable byte
BC' = first byte following the program just loaded

Procedure for Writing a Program File

Program files must have fixed-length records of length 256 and must have the P (program) access type. You set both of these when you create the file, using a creation code of 1 or 2 (see the OPEN SVC).

The TRSDOS-II program loader treats the program file as a serial byte stream, independent of the record boundaries. The loader assumes the program blocks are "back-to-back" (there are no unused bytes between blocks). As programmer, you are responsible for:

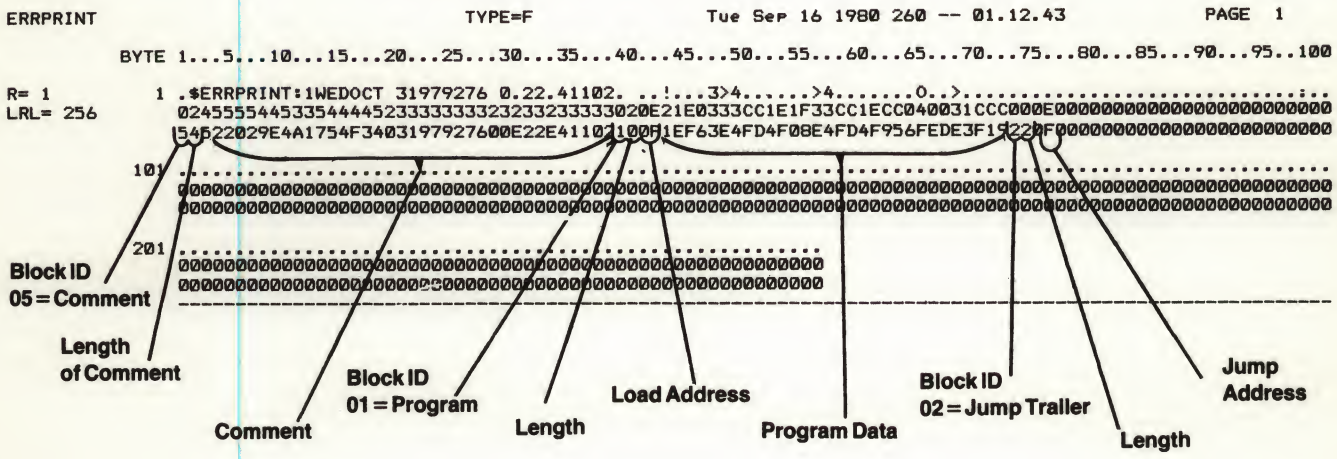
1. Storing the correct byte sequence in the logical record area.

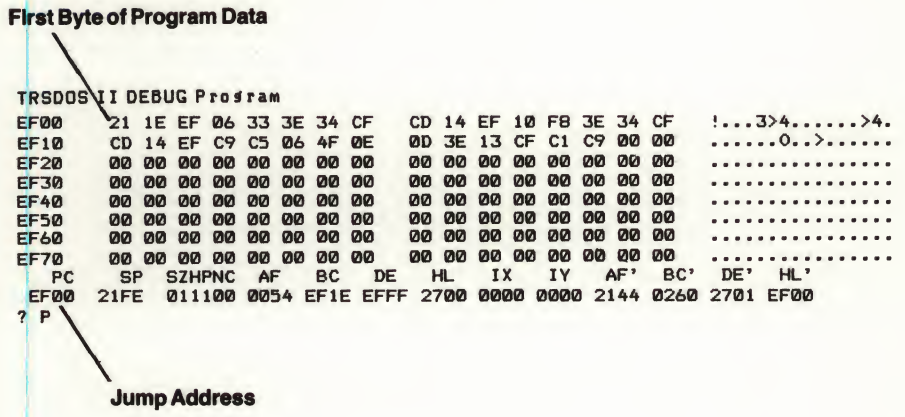
2. Calling the disk write SVC (DIRWR) when each 256-byte record is filled. Blocks can and must "span" sectors. The only unused bytes in the file are after the trailer block. See the illustration below.

TRSDOS-II does not load user program below "user RAM." You also should make sure blocks do not load above "user RAM."

Note: The TRSDOS-II DUMP command always writes a comment as the first block in the file. The comment contains the filespec, followed by the current TRSDOS-II time/date text.

Illustrated Program File Listing





High-Memory Modules: Their Addresses and Uses

As mentioned earlier, there are five high-memory modules that are conditionally loaded. They are DO, SPOOL, HOST, DEBUG and the communications drivers. They reside in high memory (above X'FØØØ'). The approximate memory map is as follows:

Module	Address
Comm drivers	X'FØØØ'
DEBUG	X'F4ØØ'
HOST*	X'F4ØØ'
SPOOL*	X'F64Ø'
DO*	X'FC8Ø'

* Cannot be used with DEBUG

Addresses may change in future releases.

If your application requires a large amount of RAM, you may need to use high memory. For example, you may have a large BASIC program or a machine-language subroutine that you must link to BASIC. In either case, you might need a full RAM for the BASIC program itself. You must use high memory. If you do use high memory, keep in mind the following:

- **If you use BASIC, the use of high memory gives you slightly less RAM for your program and variable storage than you had with TRSDOS.**

  Upon program-load, TRSDOS-II gives X'EFFF' in Register DE as the high-memory address.

  If you must, you may override this limit by specifying this command line:

  BASIC - M:upper limit of high-RAM address

  The address to use as the upper limit must never exceed decimal 63487. The same rule applies to an applications program.

- **The use of high RAM prevents the use of certain high-memory modules.**

Assume that you invoke a system function that loads high RAM with a module. Then, your application loads something on top of it. (TRSDOS-II cannot prevent you from overlaying an already loaded high-memory module.) You are going to have problems. (See the memory map above for details.)

Be careful about overlaying the addresses of the following modules:

- **Communications drivers** -- If you use a serial printer, do not overlay X'FØØØ'-X'F3FF'.

  The code in that area must be intact if your application has printed output. That is when the comm drivers are active. Overlaying in this area can cause serious problems.

- **SPOOL** -- This is another module that can be used by almost any application that has printed output. To give operator flexibility, you should not use the RAM area of X'F64Ø'-X'F8ØØ' in your application software.

  If you must use this RAM area, make it clear to the operator that he may not use SPOOL. If any of the spool code space is overlayed after TRSDOS-II has loaded the spool code in that area, serious problems will result.

- **HOST** -- Lets a remote terminal execute your applications and programs as if the remote were local (the local keyboard and video). Again, using this space limits the operator's flexibility and causes serious problems.

You can use the high-memory space without causing problems, if you plan carefully. Know what you need and when you need it. Then, tell the operator what to do (or not to do) when running your application.

For example, assume that your application program does not create printed output. In this case, you do not need the spool or comm drivers areas (if the comm drivers are needed just for serial printer support).

To use this space, make sure SPOOL and the comm drivers are not active before or during the run of the application. Do this by starting execution of the application with a DO file (which resides in the top of RAM).

In this DO file, you may turn off SPOOL and the comm drivers before the application is executed. Remember that you must turn off both Serial Channels A & B for the high-memory comm drivers to be inactive.

For full operational flexibility, you should stay away from the high-memory areas. Then, you can use all of the high-memory routines during program execution.

Future releases of TRSDOS-II may provide more features or routines that will use high memory. Keep this in mind when you design your applications. To use these features, you may have to update your high-memory subroutines. Consider the efforts required to make these program changes. Your program may be hard to update, or it may be hard to get revised copies out to the users.

ACTRL **Function Code 1ØØ**
Control Channel A

Performs control functions on Serial Channel A.

TRSDOS-II sets up ARCV, ATX, and ACTRL when you initialize Channel A with RS232C. If you call any of these routines while the channel is not initialized (active), you get an error return code of 1 (no function code exists).

Entry Conditions

A	= 1ØØ
B	= option switch

Exit Conditions

NZ	= Error
B	= status code

The option switch settings are:

option switch	Result
Ø	Gets the current status of the serial I/O (SIO) channel into Register B.
1	Gets the current character count in the serial receive buffer into B.
2	Turns on Request to Send (RTS). RTS is on when the channel is initialized.
3	Turns off RTS.
4*	Starts transmitting the break sequence.
5	Stops transmitting the break sequence.
6	Resets the serial receive buffer count to zero.
7	Resets the SIO error condition.

* Before transmitting a break sequence, make sure the sure the transmit buffer is empty. As the programmer, you are responsible for sending the break sequence for the appropriate time period, as required by the application or host computer.

The 8-bit grouping of flags returned in Register B consists of:

Bit	Meaning
Ø	Clear to Send (CTS) is not present
1	Unused
2	Transmitter is busy
3	Modem data carrier is not present
4	Parity error is occurring in the byte that is being received
5	Data overflow is occurring because of a byte that is being received
6	Framing error is occurring in the byte that is being received
7	Break sequence (extended null character) is being received

ARCV **Function Code 96**
Channel A Receive

Returns a single character from Serial Channel A.

TRSDOS-II sets up ARCV, ATX, and ACTRL when you initialize Channel A with RS232C. If you call any of these SVCs while the channel is not initialized, you get an error return code of 1 (no function code exists).

Input Buffer

Each channel (A and B) has an internal receive buffer to reduce overruns when receiving data at high speeds. This buffer is a first-in, first-out buffer (FIFO) which is established when the channel is initialized.

When a character is received, the character and the hardware status code are placed in the Channel A buffer. If Channel A has not been initialized, no characters are received.

An overrun occurs when a character is received and there is no more room in the buffer. The character that caused the overrun replaces the last character received. Bit 5 of the hardware status code is set; this indicates that an overrun occurred.

When the ARCV SVC is executed, the oldest character in the buffer is returned to Register B; the hardware status at the time the character was received is returned to Register A.

> Note: When a character that caused an error is returned to Register B, the error is indicated in Register A.

If there are no characters "waiting" in the buffer, the communications status returned in Register A reflects the current status of the serial interface.

Entry Conditions

A = 96

Exit Conditions

A	= status code
B	= character returned, if any
NZ	= Character was not received, check the status code
C flag	= Modem carrier was not present when you entered the SVC

The 8-bit grouping of flags returned in Register A (after serial input) consists of:

Bit	Meaning
Ø	Not used
1	Not used
2	Not used
3	Modem carrier was not present
4	Parity error occurred on the last character received in Register B
5	Data is lost because of an overflow; Register B contains the last character
6	Framing error occurred on the last character received
7	Break sequence (extended null character) was received

ATX **Function Code 97**
Channel A Transmit

Outputs a single character to Serial Channel A.

TRSDOS-II sets up ATX, ARCV, and ACTRL when you initialize Channel A with RS232C. If you call any of these SVCs while the channel is not initialized, you get an error return code of 1 (no function code exists).

TRSDOS-II transmits data bytes even if no carrier is present. You must check the status flags and Register A for error conditions. "Carry flag set" (C flag) means that no carrier is present during transmission. You may ignore this flag if your application does not need the carrier to be present.

Register A contains status information after serial output (upon return from ATX).

Entry Conditions

A	=	97
B	=	character to be sent, (ASCII code)

Exit Conditions

A	=	status code
NZ	=	Character is not transmitted, check the status code
C flag	=	Modem carrier was not present when you entered the SVC

The 8-bit grouping of flags returned in Register A (after serial output) consists of:

Bit	Meaning
Ø	Clear to Send (CTS) was not present
1	Not used
2	Transmitter busy
3	Modem carrier was not present
4*	Parity error on the last character received
5*	Data is lost because of a hardware-level overflow

Bit	Meaning
6*	Framing error occurred on the last character received
7*	Break sequence (extended null character) was received

* TRSDOS-II returns bits 4 through 7 only when it does not send the character.

BCTRL **Function Code 1Ø1**
Control Channel B

Performs control functions on Serial Channel B.

TRSDOS-II sets up BRCV, BTX, and BCTRL when you initialize Channel B with RS232C. If you call any of these SVCs while the channel is not initialized, you get an error return code of 1 (no function code exists).

Entry Conditions

A	= 1Ø1
B	= options switch

Exit Conditions

NZ	= Error
B	= status code

Valid option switch settings are:

option switch	Result
Ø	Gets the current status of the serial I/O channel into Register B.
1	Gets the current character count in the serial receive buffer into B.
2	Turns on Request To Send. RTS is on when channel is initialized.
3	Turns off RTS.
4*	Starts transmitting the break sequence.
5	Stops transmitting the break sequence.
6	Resets the serial receive buffer count to zero.
7	Resets the SIO error condition.

* Before transmitting a break sequence, make sure the sure the transmit buffer is empty. As the programmer, you are responsible for sending the break sequence for the appropriate time period, as required by the application or host computer.

The 8-bit grouping of flags returned in Register B consists of:

Bit	Meaning
Ø	Clear to Send (CTS) is not present
1	Unused
2	Transmitter is busy
3	Modem data carrier is not present
4	Parity error is occurring in the byte that is being received
5	Data is lost because of an overflow caused by a byte that is being received
6	Framing error occurred in the byte that is being received
7	Break sequence (extended null character) was received

BINDEC **Function Code 21**
Binary/Decimal

Converts a 2-byte binary number to an ASCII-coded decimal, and vice versa. The decimal range is from Ø to 65535.

Entry Conditions

A = 21
B = function switch
B = Ø: convert binary to ASCII decimal
B ≠ Ø: convert ASCII decimal to binary

If B = Ø (binary to decimal):
DE = 2-byte binary number to convert
(HL) = 5-byte area to contain the ASCII decimal value
The field contains decimal digits in the range [X'3Ø',X'39'], with leading zeroes on the left as needed to fill the field. For example, ØØØ21 represents 21.

If B ≠ Ø (decimal to binary):
(HL) = 5-byte area containing the ASCII decimal value to convert

Exit Conditions

(HL) = decimal value
DE = binary value

BINHEX **Function Code 24**
Binary/Hexadecimal

Converts a 2-byte binary number to an ASCII-coded hexadecimal, and vice versa. The hexadecimal range is from X'ØØØØ' to X'FFFF'.

Entry Conditions

A = 24
B = function switch
B = Ø: convert binary to ASCII hexadecimal
B ≠ Ø: convert hexadecimal to binary

If B = Ø (binary to hexadecimal):
DE = 2-byte binary number to convert
(HL) = 4-byte area to contain the ASCII hexadecimal value
The field contains hexadecimal digits with leading zeroes on the left, as needed to fill the field. For example, ØØFF represents the number X'FF'.

If B ≠ Ø (hexadecimal to binary):
(HL) = 5-byte area containing the ASCII hexadecimal value to convert

Exit Conditions

(HL) = hexadecimal value
DE = binary value

BRCV **Function Code 98**
Channel B Receive

Returns a single character from Serial Channel B.

TRSDOS-II sets up BRCV, BTX, and BCTRL when you initialize Channel B with RS232C. If you call any of these SVCs while the channel is not initialized, you get an error return code of 1 (no function code exists).

Input Buffer

Each channel (A and B) has an internal receive buffer to reduce overruns when receiving data at high speeds. This buffer is a first-in, first-out buffer (FIFO) which is established when the channel is initialized.

When a character is received, the character and the hardware status code are placed in the Channel B buffer. If Channel B has not been initialized, no characters are received.

An overrun occurs when a character is received and there is no more room in the buffer. The character that caused the overrun replaces the last character received. Bit 5 of the hardware status code is set; this indicates that an overrun occurred.

When the BRCV SVC is executed, the oldest character in the buffer is returned to Register B; the hardware status at the time the character was received is returned to Register A.

> Note: When a character that caused an error is returned to Register B, the error is indicated in Register A.

If there are no characters "waiting" in the buffer, the communications status returned in Register A reflects the current status of the serial interface.

Entry Conditions

A = 98

Exit Conditions

A	=	status code
B	=	character returned, if any
NZ	=	Character not received; check the status code
C flag	=	Modem carrier was not present when you entered the SVC

The 8-bit grouping of flags returned in Register A (after serial input) consists of:

Bit	Meaning
Ø	Not used
1	Not used
2	Not used
3	Modem carrier was not present
4	Parity error occurred on the last character received in Register B
5	Data is lost because of an overflow; Register B contains the last character
6	Framing error occurred on the last character received
7	Break sequence (extended null character) was received

BTX **Function code 99**
Channel B Transmit

Outputs a single character to Serial Channel B.

TRSDOS-II sets up BTX, BRCV, and BCTRL when you initialize Channel B with RS232C. If you call any of these routines while the channel is not initialized, you get an error return code of 1 (no function code exists).

TRSDOS-II transmits data bytes even if no carrier is present. You must check the status flags and Register A for error conditions. "Carry flag set" (C flag) means that no carrier is present during transmission. You may ignore this flag if your application does not "care" whether the carrier is present.

Register A contains status information upon return from BTX.

Entry Conditions

A	=	99
B	=	character to be sent (ASCII code)

Exit Conditions

A	=	status code
NZ	=	Character not transmitted; check the status code
C flag	=	Modem carrier was not present when you entered the SVC

The 8-bit grouping of flags returned in Register A (after serial output) consists of:

Bit	Meaning
Ø	Clear to Send (CTS) was not present
1	Not used
2	Transmitter is busy
3	Modem carrier was not present
4*	Parity error occurred on the last character received
5*	Data is lost because of a hardware-level overflow

Bit	Meaning
6*	Framing error occurred on the last character received
7*	Break sequence (extended null character) was received

* TRSDOS-II uses Bits 4-7 only when it does not send the character.

CLOSE **Function Code 42**

Close Disk Files

Ends access to the file you specify. This SVC first writes all unsaved data to the disk, then updates the directory before closing the file.

Entry Conditions

A = 42
(DE) = data control block of the currently open file

Exit Conditions

NZ = Error
A = error code

Upon return, the filespec (except for the password) automatically is put back into the DCB.

CLRXIT **Function Code 57**

Clear User Memory and Jump to TRSDOS

Clears (writes binary zeroes to) user memory and gives control to TRSDOS-II.

Entry Conditions

A = 57

Exit Conditions

None

CURSOR **Function Code 26**
Blinking Cursor

Turns the blinking cursor on or off. TRSDOS-II keeps track of the current cursor position, whether the cursor is on or off.

Entry Conditions

A = 26
B = function switch
B = Ø: blinking cursor off
B ≠ Ø: blinking cursor on

Exit Conditions

None

DATE **Function Code 45**
Return Date String

Sets or returns the time and date. TRSDOS-II returns the data as a 26-byte ASCII string with 8 fields.

Contents of Time/Date string:

Fri	Oct	1	1982	274	13.2Ø.42	1Ø	4
Name of Day	Mon.	Day of Mon.	Year	Day of Year	Time	Mon. of Year	Day of Week

Represents the data "Friday, October 1, 1982, the 274th day of the year, 1:2Ø:42 p.m., the tenth month of the year, the fourth day of the week."

Monday is considered day Ø. The date calculations are based on the Julian Calendar.

Entry Conditions

A = 45
B = function switch
B = Ø: get time/date
B = 1: set date
B = 2: set time
(HL) = a dependent of the contents of Register B

If B = 1:
(HL) = 1Ø-byte buffer containing the date (The date is in the form mm/dd/yyyy.)

If B = 2:
(HL) = 8-byte buffer containing the time (The time is in the form hh.mm.ss.)

Exit Conditions

NZ = Error
If B = Ø:
(HL) = 26-byte buffer where the time/date are to be stored

DELAY **Function Code 6**
Delay Loop

Provides a delay routine and returns control to the calling program after the time you specify has elapsed.

Background processing, such as the SPOOL printing function, uses this delay time.

Entry Conditions

A = 6
BC = delay multiplier switch
BC = Ø: the delay time is 426 milliseconds
BC ≠ Ø: the delay time is:
6.5 * (BC - 1) + 22 microseconds.

Exit Conditions

None

DIRRD **Function Code 35**
Direct-Read

Reads the specified record of an FLR file.

Also, you can use DIRRD to read the first record or the EOF (X'FFFF') of a VLR file.

Upon return, your record is in the record area pointed to by RECADR in the parameter list. Or, if the record length is 256, and you are reading a fixed-length record, it is in the area pointed to by BUFADR (see the OPEN SVC for more information).

Entry Conditions

A = 35
(DE) = data control block of the currently open file
BC = record number
BC = Ø: position to beginning of file
BC = X'FFFF': position to end of file
HL = Reserved

If you chose the E mode when you opened the file, then:

BC = address of 3 bytes in RAM The format is:

upper byte	middle byte	lower byte

where ØØ ØØ ØØ = first record of file
and FF FF FF = end of file (EOF)

Exit Conditions

NZ = Error
A = error code

DIRSET **Function Code 59**
Directory Information

Gets directory information on any open file.

DIRSET sets up the 8-byte block of memory that is one of the entry parameters for the RDDIR SVC. After you open the file, you can:

1. Find out which drive contains the file (See the RDDIR SVC).

2. Set up the 8-byte block in user memory (BC) for RDDIR. This lets RDDIR get directory information about the open file.

Entry Conditions

A = 59
DE = address of the open data control block
BC = address of the 8-byte block of user memory

Exit Conditions

All registers return unchanged, except Register A.

BC = address of the 8-byte block of user memory set for RDDIR call to get the directory information of the open data control block (pointed to by DE)
Z = No error
NZ = I/O error
A = error code

Note: Be sure to load Register Pair DE with zero (LD DE,Ø) after executing DIRSET and before executing RDDIR. If DE is non-zero, RDDIR expects a wildcard mask and, in most cases, cannot find a match.

DIRWR **Function Code 44**
Direct-Write

Writes a specified record to a specified position in an FLR file. Also, you can use DIRWR to write the first record (Ø) or the EOF (X'FFFF') record in a VLR file.

Before calling DIRWR, put your record into the record area pointed to by RECADR in the parameter list. Or, if the record length is 256 and the record type is fixed, put the record in the area pointed to by BUFADR (see the OPEN SVC for more information).

Entry Conditions

A = 44
(DE) = data control block of the currently open file
BC = record number
BC = Ø: position to beginning of file
BC = X'FFFF': position to end of file
(HL) = Reserved

If you chose the E mode when you opened the file, then:
BC = address of 3 bytes in RAM The format is:

upper byte	middle byte	lower byte

where ØØ ØØ ØØ = first record of file
and FF FF FF = record after current EOF (EOF +1)

Exit Conditions

NZ = Error
A = error code

DISKID **Function Code 15**
Read Disk ID

Reads the diskette ID(s) from any or all of Drives Ø-7. This is useful when the program must ensure that the operator has inserted the proper diskette.

TRSDOS-II places the diskette ID(s) in the buffer pointed to by Register Pair HL. If a drive is not ready, TRSDOS-II places blanks in the buffer. If you specify 255, TRSDOS-II reads the IDs from all drives. If you have do not have hard disks, the last 32 bytes of the buffer are blank.

Entry Conditions

A = 15
B = drive select code
B = Ø-7: read from specified drive (Ø-7)
B = 255: read all disks' IDs
(HL) = disk ID buffer

If B = Ø-7:
The buffer must be 8 bytes long.

If B = 255:
The buffer must be 64 bytes long.

Exit Conditions
(HL) = disk ID(s)
NZ = Error
A = error code

DOSCMD **Function Code 37**
Execute TRSDOS-II Command

Passes a command string to TRSDOS-II Ready mode for execution. After the command is executed, control returns to TRSDOS-II. All open files are closed automatically.

Entry Conditions

A = 37
(HL) = command string
B = length of string

Exit Conditions

None

ERRMSG **Function Code 52**
Error Message

Returns an 8Ø-byte error description to the specified buffer area, in response to the specified error number.

Entry Conditions

A	=	52
B	=	error number
(HL)	=	8Ø-byte buffer area above X'27FF'

Exit Conditions

NZ	=	Error
A	=	error code

ERROR **Function Code 39**
Display Error Number

Displays the message ERROR, followed by the specified error code, at the current cursor position.

Entry Conditions
A = 39
B = error number

Exit Conditions
NZ = Error
A = error code

FILPTR **Function Code 58**
Get Pointers of a Open File

Gives information on an open user file. If it does not make a match, the SVC returns an error.

Note: We recommend restricting your use of the FILPTR since this SVC is restricted to 96 files. If you are familiar with FILPTR, you should change your existing programs to use RDDIR instead.

Entry Conditions

A	= 58
(DE)	= data control block of the currently open file

Exit Conditions

Z flag	= No error
NZ	= Error
B	= drive that contains the file (binary Ø-7)
C	= position of the file in the diskette directory (binary 1-96)

Notes

This SVC is intended for use with the RAMDIR SVC. After opening a file, you may:

1. Use FILPTR to find out which drive contains the file and which directory record contains the file's information.

2. Use RAMDIR to obtain information about that file (current file space allocated/used, protection level, and so on).

HLDKEY **Function Code 29**

Enable the <HOLD> Key

Suspends and restarts terminal output whenever you press <HOLD>.

This SVC must first be enabled; call the SVC with B = 1. You must periodically call the SVC to see if the <HOLD> key has been pressed; call the SVC with B > 1. This places the program in control of the pause checking. If <HOLD> has been pressed, the program pauses until <HOLD> is pressed again.

Returning to the TRSDOS-II READY command level turns off the <HOLD> processor. However, you can execute a command without turning off the <HOLD> processor. See the JP2DOS and DOSCMD SVCs for ways to execute a command.

Some library commands and utilities, including DIR and LIST, reset the <HOLD> processor.

No matter how many times you press <HOLD> before you call HLDKEY, TRSDOS-II interprets it as one keystroke, thus starting the pause.

Entry Conditions

A = 29
B = function code

The HLDKEY function codes are:

Contents of B	Meaning
Ø	Turns off the <HOLD> processor. Pressing <HOLD> generates X'ØØ'.
1	Turns on the <HOLD> processor. Pressing <HOLD> does not generate keyboard data. TRSDOS-II intercepts it.
>1	Checks for the <HOLD> key. If you have pressed it, TRSDOS-II pauses until you press it again.

Exit Conditions

NZ = <HOLD> processor is off

INITIO **Function Code Ø**
Initialize I/O

Initializes all input/output drivers.

This SVC is already done by TRSDOS-II. You do not need to call it, except in extreme error conditions.

Entry Conditions
A = Ø

Exit Conditions
NZ = Error
A = error code

JP2DOS **Function Code 36**
Jump to TRSDOS-II

Returns control to TRSDOS-II, after closing all open files and doing system housekeeping.

Another way to do this is to execute an RST Ø instruction. There are no entry conditions for this method.

Entry Conditions
A = 36

Exit Conditions
None

KBCHAR **Function Code 4**
Keyboard Character

Gets one character from the keyboard. If characters are present in the key-ahead buffer, the first character in the buffer is returned to Register B.

The <BREAK> key is masked from you. It is never returned, since it is intercepted by TRSDOS-II. If you enable a SETBRK routine, control passes to the processing program whenever <BREAK> is pressed (see the SETBRK SVC). Otherwise, control passes to TRSDOS-II READY.

Entry Conditions

A = 4

Exit Conditions

B = character found (if any)
If no character is returned, B is unchanged.
NZ = No character is present
A = error code

KBINIT **Function Code 1**
Keyboard Initialize

Initializes the keyboard input driver. You might want to call KBINIT before starting keyboard input, to clear previous keystrokes and the key-ahead buffer. TRSDOS-II automatically does this at startup.

Entry Conditions
A = 1

Exit Conditions
NZ = Error
A = error code

KBLINE **Function Code 5**
Keyboard Line

Inputs a line from the keyboard to a buffer and echoes the line to the display, starting at the current cursor position. As it receives and displays each character, KBLINE advances the cursor to the next position.

When you enter KBLINE, the input buffer contains a string of periods, which it sends to the display. This string is for your convenience. It indicates the length of the input field.

The keyboard line ends when you press <ENTER> or when the input buffer is filled. When you end the line input, KBLINE sends a carriage return and an "erase to end of screen" command to the display. It stores a carriage return as a character input only if you press <ENTER>.

Entry Conditions

A = 5
(HL) = input buffer
B = maximum number of characters to receive (1-255)

Exit Conditions

B = actual number of characters input (including carriage return)
C = terminating character of input
C = X'ØØ': input buffer is filled without a carriage return
C = X'ØD': line ended with a carriage return

If you type a control code not listed below, KBLINE places it in the buffer and represents it on the display with the ± symbol.

Key	Hex Code	Function
<BACKSPACE>	Ø8	Backspaces the cursor and erases a character.
<ENTER>	ØD	Ends the line. Clears trailing periods on the display but not in the buffer.

Key	Hex Code	Function
<CTRL> <W>	17	Fills the remainder of the input buffer with blanks. Blanks the remainder of the display line.
<CTRL> <X>	18	Fills the remainder of the input buffer with blanks. Blanks to the end of the display.
<ESC>	1B	Reinitializes the input function by filling the input buffer with periods and restoring the cursor to its original position.
<left arrow>	1C	Backspaces the cursor to allow editing of the line. Does not erase characters.
<right arrow>	1D	Advances the cursor to allow editing of the line. Does not erase characters.

KBPUT **Function Code 3Ø**
Put Character into Key-Ahead Buffer

Puts one character into the keyboard key-ahead buffer, in the same manner that pressing that key on the keyboard does.

The character is placed after any characters already in the buffer.

If the character is X'ØØ', TRSDOS-II puts it into the buffer if HLDKEY is off. If HLDKEY is on, X'ØØ' triggers hold processing. (See the HLDKEY SVC.)

If the character is X'Ø3', it invokes the <BREAK> key processing. (See the SETBRK SVC.)

Entry Conditions

A	=	3Ø
B	=	character to be put into the buffer

Exit Conditions

Z	=	Character is put into the buffer
NZ	=	Buffer is full; the character is not in the buffer
A	=	error code

KILL **Function Code 41**
Delete File From Directory

Deletes the specified file from the directory. You must close a file before you can KILL it.

Entry Conditions

A	=	41
(DE)	=	data control block

Exit Conditions

NZ	=	Error
A	=	error code

LOCATE **Function Code 33**
Locate Record

Returns the number of the current record (the last record accessed). You can use LOCATE only with FLR files.

Entry Conditions

A = 33
(DE) = data control block of the currently open file
HL = Reserved

Exit Conditions

BC = current record number
NZ = Error
A = error code

If you chose the E mode when you opened the file, then:

BC = address of 3 bytes in RAM where the record number is to be stored, in the following format:

upper byte	middle byte	lower byte

where ØØ ØØ ØØ = first record of file

LOOKUP **Function Code 28**
Look Up in a Table

Does a lookup function on a table of 3-byte entries. Each entry looks like this:

Byte 1	Byte 2	Byte 3
Search Key	Data, e.g., address in LSB, MSB sequence	

At the end of the table, you must place a 1-byte X'FF'. Since X'FF' is the table terminator, your search keys must be between X'ØØ' and X'FE'.

Given a 1-byte search argument, LOOKUP locates the first matching entry in the table. It uses a sequential search algorithm.

If the table contains search keys followed by addresses, then (upon return from the routine with the Z flag set) you can JP (HL) to transfer control to the desired address.

Entry Conditions

A = 28
(HL) = byte 1 of the table
B = search argument

Exit Conditions

NZ = Search argument is not found
HL = data
H contains byte 3
L contains byte 2

MPYDIV **Function Code 23**
Multiply Divide

Multiplies or divides with one 2-byte value and one 1-byte value.

Entry Conditions

```
     A      = 23
     B      = function switch

If B = Ø:  multiply
     HL     = multiplicand
     C      = multiplier

If B ≠ Ø: divide
     HL     = dividend
     C      = divisor
```

Exit Conditions

```
If B = Ø:
     HL     = product (HL * C)
     C flag = Overflow occurred
     A      = overflow byte
     Z      = Product is Ø

If B ≠ Ø:
     HL     = quotient (HL/C)
     C      = remainder
     C flag = Dividing by Ø; division is not attempted
     Z      = Quotient is Ø
```

OPEN **Function Code 4Ø**
Open File

Creates new files and opens existing files.

A given file can be open under only one data control block (DCB) at a time. Because of the versatile file processing routines, this one DCB is enough to handle the various I/O applications.

Entry Conditions

A = 4Ø
(DE) = data control block
(HL) = parameter list

Exit Conditions

NZ = Error
A = error code

Before calling OPEN, you must reserve space for the data control block, parameter list, buffer area, and record area.

Data Control Block (6Ø bytes)

The data control block (DCB) is used by TRSDOS-II for file access bookkeeping. Use it specify the filespec when opening a file.

Before calling OPEN, place the filespec at the beginning of the DCB, followed by a carriage return. (See "File Specification" in the "Introduction.")

For example:

Contents of First Bytes of DCB Before Open
($ signifies a carriage return)

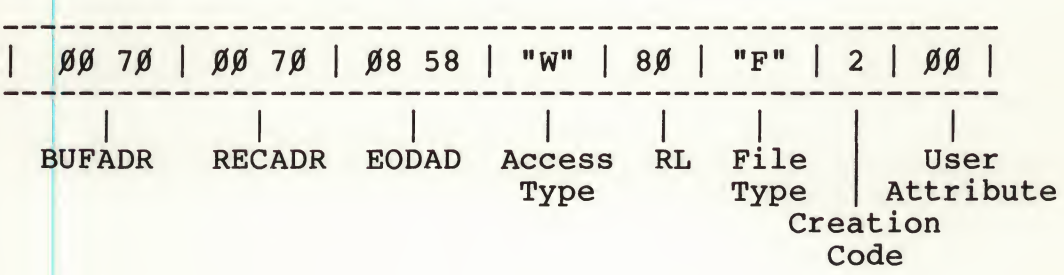
f i l e n a m e/e x t . p a s s w o r d :d (disk name) $

While a file is open, TRSDOS-II replaces the filespec with information TRSDOS-II uses for bookkeeping. When you close the file, TRSDOS-II returns the original filespec (except the password) to the DCB.

Never modify any part of the DCB while the file is open. The results are unpredictable.

Parameter List (11 bytes)

Contents of Parameter List (sample)



BUFADR (Buffer Address)

Two-byte field that points to the beginning of the buffer area.

The buffer area is the space TRSDOS-II uses to process all file accesses. When spanned records are required, you must reserve 512 bytes. When spanning is not required, reserve only 256 bytes.

With FLR files, spanning is required only when the record length is not an even divisor of 256. For example, if the record length is 64, then each physical record contains four records, and no spanning is required. In this case, reserve only 256 bytes for processing.

However, if the record length is 24 (not an even divisor of 256), then some records must be spanned. In this case, reserve 512 bytes.

With VLR files, you must reserve 512 bytes for processing. Spanning may be required, depending on the lengths of the records.

RECADR (Record Address)

Two-byte field that points to the beginning of the record area.

This is where TRSDOS-II places the record after a disk read and where you would put the record to be written to disk.

For FLR files with a record length of 256, RECADR is not used. Your record is in the buffer area pointed to by BUFADR.

If you have an FLR file with a record length not equal to 256, make this buffer the same length as the record.

If you have VLR files, be sure to make the buffer long enough to contain the longest record in the file (including the length byte). If you are not sure of the length, reserve 256 bytes.

EODAD (End of Data Address)

Two-byte field which can be used to give TRSDOS-II a transfer address to use in case TRSDOS-II reaches the EOF during an attempted read.

Control transfers to the EODAD if this happens. If EODAD = Ø and the EOF is reached, the SVC returns with the EOF error code in Register A.

Access Type

Specifies the type of access you desire. It can be any of the following:

ASCII Character	Meaning
R	Read only
W	Read/Write (data files)
P	Read/Write access to Z8Ø program files

If OPEN finds a program file, TRSDOS-II returns a P in the read/write field in the parameter list. For example, a call to OPEN with a parameter list of:

ØØ7Ø ØØ7Ø Ø858 R 8Ø F Ø ØØ

returns the following, if the file found is a program file:

```
ØØ7Ø ØØ7Ø Ø858 P 8Ø F Ø ØØ
```

RL (Record Length)

One-byte field that specifies the record length to be used.

Zero indicates a record length of 256. For VLR files, ignore this field. If the file exists, and the creation code is Ø, TRSDOS-II supplies the correct RL value, regardless of what is stored there.

File Type

One-byte field that contains an ASCII "F," "V," or "E" (for "fixed-length," "variable-length," or "extended mode").

Once a file exists, this attribute cannot be changed. If the file exists, and the creation code is Ø, TRSDOS-II supplies the correct F, V or E value.

If you specify E, then DIRRD, DIRWR and LOCATE use a 24-bit, indirect, 3-byte number. The E specifies FLR records and an extended mode of specifying record numbers. The directory shows F, even though you specify the E mode.

Creation Code

One-byte field that contains a binary number "Ø," "1," or "2."

Code	Meaning
Ø	Opens the existing file, but does not reset the record length and EOF.
1	Creates a new file. Returns an error if a file of the same name already exists on the specified drive. Sets the record length and EOF at OPEN.

Code	Meaning
2	Opens a file, overwriting any existing file of the same name. Otherwise, it creates a new file. Resets the record length and EOF.

User Attribute

Lets you give your own ID number to certain types of data files. You can use any number from 32-255.

TRSDOS-II does not examine this user attribute. It is solely for your convenience.

You can assign this user attribute only to files you open with a creation code of 1 or 2. Files opened with a creation code of Ø retain the file's previously assigned user attribute. Any file created with the CREATE command has a user attribute value of Ø.

PARSER **Function Code 46**
Analyze Text Buffer

Analyzes or "parses" a text buffer into fields and subfields.

PARSER is a convenient and powerful SVC. It is useful for analyzing TRSDOS-II command lines, including keyword commands, file specifications, keyword options and parameters. You can use it also as a fundamental routine for a compiler or text editor.

Entry Conditions

A = 46
(HL) = text buffer
(DE) = list address block
DE = Ø: no lists are to be used
C = maximum length of the parse

Exit Conditions

(HL) = field position
If a delimited field was found:
(HL) = first byte of the field
If no field was found:
(HL) = terminator or non-blank separator
If the parse reached the maximum length:
(HL) = last byte of the buffer
B = actual length of the field (excluding leading and trailing blanks)
A = character preceding the field just found
If B = Ø, A = X'FF'.
C = number of bytes remaining to parse after the terminator or separator (trailing blanks have been parsed).
D = separator or terminator at the end of the field
If D = X'FF', then the parse stopped without finding a non-blank separator or terminator.
E = displacement pointer for the next parse call

Add the displacement pointer (E) to the field position (HL) to get:
a) Beginning address of the next field, or
b) Address of the byte following the last byte parsed

If the parse reached the maximum length:
E + HL = address following the end of the text buffer

If the parse did not reach the maximum length, and E = 1:
E + HL = address following the separator or terminator

If the parse did not reach the maximum length:
Z flag = Parse ended with a separator
NZ = Parse ended with a terminator
C flag = There were trailing blanks between the end of the field and the next non-blank separator or terminator
NC = There were no trailing blanks

PARSER is designed to allow repetitive calls for processing a text buffer. On exit from the parse, parameters for the next call are all readily available in the appropriate registers.

Field

A field is any string of alphanumeric characters (A-Z, a-z, Ø-9) with no embedded blanks, delimited by separators and terminators.

Separators and Terminators

Separators and terminators are used to delimit (or separate) fields within a text buffer. PARSER has pre-defined sets of field-characters and separators. You either can use these or re-define them to suit your application.

For example, the line:

BAUD=3ØØ,PARITY=EVEN WORD=7

contains 6 fields: BAUD, 3ØØ, PARITY, EVEN, WORD, and 7.

A separator is any non-alphanumeric character. PARSER stops when it encounters a separator, except when the separator is a blank (X'2Ø'). PARSER ignores leading and trailing blanks. After the trailing blanks, the parse stops at the beginning of the next field or at the first non-blank separator.

You can also define terminators, which stop PARSER regardless of whether it has found a field. Unless you specifically define these terminators, the parse stops only on non-blank separators.

Separators and terminators have the same effect on a parse. The only difference is in how they affect the F (Flag) register on exit.

A field can be delimited by paired quotation marks, also:

"field" or 'field'

When you use quotation marks, all characters (not just alphanumerics) are taken as part of the field. The quotation marks are not included in the field. For example:

'DATE(Ø7/11/79)'

is interpreted as one field that contains everthing inside the quotation marks.

When you use a quotation mark to mark the start of a field, you must use the same kind to end the field. This lets you use quotation marks within a field. For example:

"X'FFØØ'"

is parsed as one field that contains everything inside the double quotes, including the single quotes.

To Re-define the Field, Separator, and Terminator Sets

If you need to change the field and separator sets, or define terminators, you can provide three change-lists via a list address block.

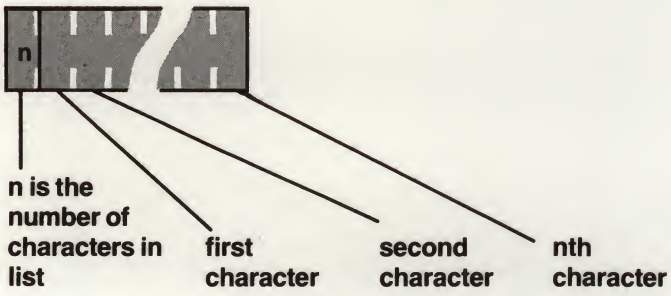
List Address Block

The list address block is six bytes long and contains three 2-byte addresses (LSB,MSB), one for each change-list:

List 1: characters to be used as terminators
List 2: additions to the set of field characters

List 3: deletions from the set of field characters (alphanumerics to be interpreted as separators)

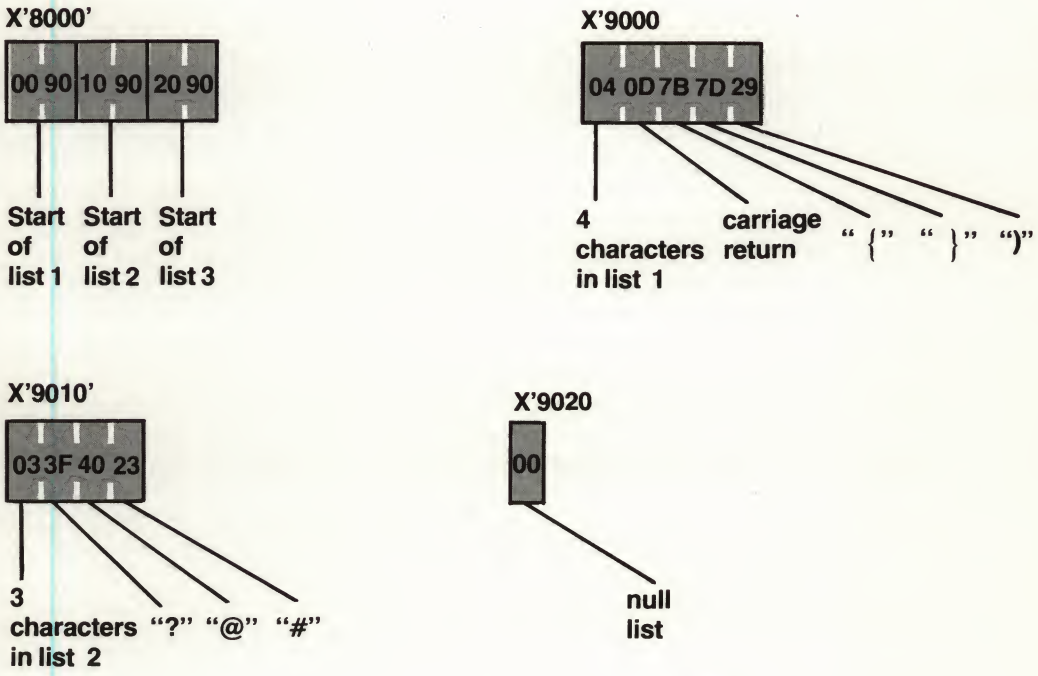
Each list has the following form:



Notes

- There are three ways to indicate a null list:
  a) Set the character-count byte (n) equal to zero.
  b) Set the pointer in the list address block to zero.
  c) Set DE equal to zero, if you aren't going to provide any lists.
- Characters are stored in lists in ASCII form.
- If a character appears in more than one list, it has the characteristics of the first list that contains it.

Here is a typical list address block with its associated lists. Assume that on entry to PARSER, DE = X'8ØØØ'.



Sample Programming

The following code shows typical repetitive uses of PARSER to break up a parameter list.

```
;--------------PREPARE FOR PARSING LOOP-----------------------------
        LD      C,MAXLEN        ; C = Maximum length to parse
        LD      E, 0            ; For initial call to NXTFLD
        LD      HL,BUFFER       ; (HL) = string to parse
;--------------PARSE LOOP-------------------------------------------
PARSE   CALL    NXTFLD          ; Routine to call PARSER
        CALL    HANDLR          ; Routine to handle new field
        JR      NZ,NXTRTN       ; Go to next routine if
                                ;  parsed ended on terminator
        LD      A,C             ; Else get new max length
        OR      A               ; Is it zero?
        JR      NZ,PARSE        ; If not, then continue
        LD      A,0FFH          ;
        CP      D               ; If D=0FFH then no separator
                                ;  at end of buffer.
        JR      Z,ERR           ; So go to error routine
        JR      NXTRTN          ; Else, then do next routine.
;--------------FIELD-HANDLING ROUTINE-------------------------------
HANDLR PUSH     AF              ; Must save status registers
                                ; and any other registers
                                ; will be changed.
                                ;
; Processing code goes here...
        POP     AF              ; Restore AF (and other
                                ; registers saved at entry)
        RET
;--------------CALL TO PARSER---------------------------------------
NXTFLD LD       D,0             ; Zero msb of DE
        ADD     HL,DE           ; (HL) = where to start parse
        LD      DE,LAB          ; (DE) = List address block
                                ; If DE=0 then no lists used.
        LD      A,46            ; Function Code
        RST     8
        RET
;--------------PROGRAM CONTINUES HERE-------------------------------
NXTRTN EQU      $
```

PRCHAR **Function Code 18**
Print Character

Sends one character to the printer.

Note: Most printers do not print until their buffer is filled or they receive a carriage return. (See your printer's manual for information.)

Normally, TRSDOS-II intercepts certain codes and does not send them directly to the printer. There are several ways to override some or all of these character translations. (See the PRINIT and PRCTRL SVCs.)

While the serial printer option is selected, allowing printer output to Channel B, TRSDOS-II recognizes two characters from Channel B. These affect all serial printer output operations:

ASCII Name	Hex Code	Result
DC3, <CTRL> <S>	13	Pauses printing
DC1, <CTRL> <Q>	11	Resumes printing

Entry Conditions

A = 18
B = ASCII code for the character to send

Exit Conditions

NZ = Error
A = error code

INTERCEPTED CODES

ASCII Name	Hex Code	Result to Printer
Tab	Ø9	Sends one to eight spaces to provide the tab function.
Vertical Tab	ØB	Same as form feed below.
Form Feed	ØC	Sends enough carriage returns or line feeds to advance the paper to the next "top of form."
Carriage	ØD	Interpreted as a line feed when the current line is empty (no characters are printed since the last carriage return or line feed). This interpretation enables the correct operation of Radio Shack printers. In the auto line-feed mode, X'ØA' is sent after every X'ØD'.
Special	8D	Sends a carriage return to the printer. In the auto line-feed mode, using this code lets you send a carriage return **without** a line feed.

PRCTRL **Function Code 95**
Control Printer Operations

Gives you various printer options and lets you check the status of printer-related functions.

If you use SPOOL's capture function, TRSDOS-II sends the captured data directly to the capture file -- regardless of which control settings you select. If you use SPOOL's print function, TRSDOS-II intercepts the printed data to the selected control settings. (See the SPOOL command for details.)

By switching between the serial and parallel output modes, you can operate with two printers. One printer can use the automatic control features of TRSDOS-II, such as the auto form feed, while the other printer is controlled by the program. This is necessary because only one set of control counters is maintained.

Entry Conditions

A	= 95
C	= option value (used with option codes 3 and 4)
B	= option code

Exit Conditions

Z flag	= No error
NZ	= Error
B	= physical page length
C	= logical page length
D	= logical line length
E	= printer option E = P: parallel printer E = S: serial printer
H	= number of characters printed on the present line
L	= number of lines printed on the present page

(See the PRINIT SVC for details on physical and logical page length and logical line length.)

The option codes are:

Code	Option
Ø	Gets printer status only (see "Exit Conditions")
1	Selects the serial printer driver (you must initialize Channel B first)
2	Selects the parallel printer driver
3	Resets the current line count to the value contained in Register C
4	Resets the current character count on current line to the value contained in Register C
5	Begins the transparent mode
6	Ends the transparent mode
7	Begins the dummy mode
8	Ends the dummy mode
9	Begins the auto line-feed (after carriage return) mode
1Ø	Ends the auto line-feed (after carriage return) mode

Explanation of Options

Serial/Parallel Printer Option

While the serial printer option is selected, allowing printer output to Channel B, TRSDOS-II recognizes two input characters from Channel B. These affect all serial printer output operations:

ASCII Name	Hex Code	Result
DC3, <CTRL> <S>	13	Pauses printing
DC1, <CTRL> <Q>	11	Resumes printing

Line Count/Character Count

TRSDOS-II maintains a single line counter and a single character counter. It updates these for either serial or parallel printing. TRSDOS-II does not maintain separate counters for serial and parallel printing.

Transparent Mode

The transparent mode overrides all data translation. All data bytes go directly to the printer. TRSDOS-II does not examine the contents or update the line and character counts.

Normally, TRSDOS-II "intercepts" certain control characters and interprets them. In this way, TRSDOS-II provides printer-related features which may not be available from the printer.

For example, tabs (X'Ø9') are intercepted so that TRSDOS-II can send the appropriate number of spaces to provide the tab function. Form feeds (X'ØC' or X'ØB') are intercepted so that TRSDOS-II can advance the paper to the next top of form.

Special settings of the printer initialization values can override individual code translation. (See the PRINIT SVC for details.)

Dummy Output Mode

The dummy mode "throws away" all printer output and returns with a "good" (Z flag set) return code. During dummy mode operation, the line and character counts remain unchanged.

Auto Line Feed

Normally, the TRSDOS-II printer driver does not output line feeds after carriage returns, because most Radio Shack printers do an automatic line feed after every carriage return.

If your printer does not do automatic line feeds after carriage returns, you may enable the TRSDOS-II auto line-feed function.

While the function is enabled, TRSDOS-II outputs a line feed after each carriage return. This is true in all modes including the transparent mode.

Note: In the auto line-feed mode, you may send a carriage return with no line feed by outputting the code X'8D'. If the printer is capable of overprinting, this code returns the carriage without advancing the paper.

Precedence of Options

The priority of the available options is (in descending order):

SPOOL capture mode (See the SPOOL command)
Dummy mode
Auto line-feed (after carriage return) mode
Transparent mode
Normal mode

PRINIT **Function Code 17**
Printer Initialization

Initializes the printer driver.

It does **not** advance the printer paper and does **not** check the printer status or availability. It operates even if the printer is off-line.

Entry Conditions

A = 17
B = physical page length
C = logical page length
D = logical line length

Exit Conditions

NZ = Error

The physical page length is the number of lines that can be printed on one page. It is normally 66.

The logical page length is the number of lines you want printed on each page. It must be less than or equal to physical page length. After that number of lines is printed, TRSDOS-II sends a top-of-page request to the printer (form feed).

The logical line length is the maximum number of characters to be printed on each line. Once that number is reached, the line is printed and a new line is begun (wrap-around).

Notes

- If the logical page length = the physical page length and both are non-zero, TRSDOS-II does not do an end-of-page skip. It translates form feeds into the number of line feeds necessary to get to the top of the next page.

- If the logical page length or the physicial page length is zero, the other also must be zero. In this case, TRSDOS-II does not translate form feeds (X'ØC') and vertical tabs (X'ØB') into line feeds, but sends them directly to the printer.

- If the logical line length is zero, then TRSDOS-II does not translate tab characters (X'Ø9') into spaces, but sends them directly to the printer. TRSDOS-II maintains an internal character count, with tabs counting as one character.

- On exit, TRSDOS-II automatically resets the current line and character counts to zero.

- TRSDOS-II assumes the printer is at the top of the form when you execute PRINIT.

- TRSDOS-II resets the dummy and transparent modes to the normal mode. Auto line feed and DUAL are unaffected by PRINIT.

- When you power up or reset TRSDOS-II, it assumes the following defaults:

Parameter (PRINIT register)		Value
physical page length (B)	=	66
logical page length (C)	=	6Ø
logical line length (D)	=	132

The other options set during initialization are:

Auto line-feed mode	=	off
Dummy mode	=	off
Transparent mode	=	off
Parallel printer	=	on

PRLINE **Function Code 19**
Print Line

Sends a line of characters to the printer's buffer. The line can include control characters as well as printable data.

See the PRCHAR SVC for a list of intercepted codes.

Entry Conditions

A = 19
(HL) = start of the buffer containing data and controls to send to the printer
B = line length (the number of characters to send)
C = control character (any character) to send after the last character in the buffer

Exit Conditions

NZ = Error
A = error code

RAMDIR **Function Code 53**
Get Disk Directory/Free Space into RAM)

Lets you examine a disk directory entry or the first 96 files in the directory, or it lets you determine the diskette's free space. The information is written into a RAM buffer.

Only non-system files are included in the RAM directory.

Note: We recommend restricting your use of the RAMDIR since this SVC is restricted to 96 files. If you are familiar with RAMDIR, you should change your existing programs to use RDDIR instead.

Entry Conditions

A = 53
B = drive number (binary Ø-7)
C = function switch

Contents of C	Result
Ø	Gets the entire directory into RAM in the format described below.
1-96	Gets one specified directory record (if it exists) into RAM in the format described below.
255	Gets free-space information in the format described below.

(HL) = buffer area

If C = Ø, compute the buffer size by this formula:
Number of bytes required = 34 * (# of non-system files + 1)

Because RAMDIR can read no more than 96 non-system files, the largest possible directory needs a buffer size of 3265 bytes.

If C = 1-96, the buffer must be 34 bytes long.
If C = 255 (free-space info desired), the buffer must be 4 bytes long.

Exit Conditions

Z flag = No error; (HL) = directory or free-space information
NZ = Error (Register A contains the TRSDOS-II error code)

RAM Directory Format

The directory is given in "records," one per file. Each record has the following form:

Byte Number	Contents	Comments
1	":"	ASCII colon marks beginning of each directory record in RAM.
2-16	filename/ ext:d <ENTER>	<ENTER> represents a carriage return. Trailing blanks are added as needed to fill 15 bytes.
17	"F" or "V"	Indicates fixed- or variable-length records.
18	LRL	Logical record length 1-byte binary Ø-255. Ø implies LRL =256 or VLR file.
19	# of extents	One-byte binary Ø-16, Null files=Ø.
2Ø-21	# of sectors allocated	Binary Ø-65535 in LSB-MSB sequence; null files = ØØ.
22-23	# of sectors in storage of data	Binary Ø-65535 in LSB-MSB sequence; null files = ØØ.
24	EOF byte (position of the first byte of the last record written)	Binary Ø-255; Ø implies the first byte in the last sector.
25-26	# of records written	Binary Ø-65535 in LSB-MSB sequence; null files = ØØ.
27	User attribute byte	Assigned when you created the file.

Byte Number	Contents	Comments
28	Protection level	Binary Ø-7.
29-31	Date created	Binary year, month, day.
32-34	Date last updated	Binary year, month, day.

When an entire directory is given (C = Ø on entry), a number sign (#), instead of the expected colon (:), marks the end of the directory. When C = from 1 to 96, the record always ends after the 34th byte, without a trailing number sign.

Free-Space List Format

Byte Number	Contents	Comments
1-2	# of free granules	*Binary LSB-MSB sequence
3-4	# of extents	Binary in LSB-MSB sequence

* The number of sectors equals the number of granules multiplied by 5.

RANDOM **Function Code 2Ø**

Random 1-Byte Value Return

This routine returns a random 1-byte value.

TRSDOS-II uses the time/date values in computing the random number. This method lessens the possibility of repetition.

Pass the routine a limit value. If the limit value is greater than 1, the value returned is in the range [Ø,(limit - 1)]. For example, if the limit is 255, TRSDOS-II returns a value in the range [Ø,254]. If the limit value is Ø or 1, the value returned is Ø.

Entry Conditions

A = 2Ø
B = limit value

Exit Conditions

C = random number
NZ = Error

RDDIR **Function Code 32**
Reads in Directory Record

Reads in one directory record from one drive at a time. It also lets you use a wildcard mask. TRSDOS-II puts the directory record into a 128-byte ASCII string in user memory.

To read the directory of a specific file, use the filespec as the wildcard mask. (Do not use the asterisk.) RDDIR then searches for the file.

(See the DIRSET SVC for details on open file directory information.)

Entry Conditions

A = 32
BC = address of the 8-byte block in user memory defined as:

binary drive #	directory index*						**
	FF	FF	FF	FF	FF	FF	ØØ

* = FF FF FF FF FF FF sets the index to the beginning of the directory
** = end-of-list terminator

DE = wildcard mask For example:

```
MASK    DEFM    '*/BAS'
        DEFB    ØDH  ;must end with
                     <ENTER>
```

If DE = Ø, no wildcard selection takes place.

HL = address of the 128-byte area where data is to be placed

Exit Conditions

HL = address of the 128-byte area where data is placed. It is in the TRSDOS-II DIR format.
Z = No error
C,NZ = Error; EOF encountered
NC,NZ = I/O error
A = error code

Offset Table for HL Buffer

Offset	Description	# Bytes
Ø	Length byte	1
1-12	Filename	12
13	File-left-open marker	1
16-23	Date created	8
25-32	Date updated	8
39-42	Attrib	4
45	File type	1
48-51	Record length	4
53-61	Number of records	9
64-7Ø	Sectors allocated	7
73-79	Sectors used	7
8Ø-127	Reserved	48

Example

```
LOOP    LD      BC,CTLBLK
        LD      DE,MASK
        LD      HL,IMAGE

        LD      A,32
        RST     8
        JR      NZ,END          ;  END OR ERROR HAS OCCURRED

        (display "IMAGE")

        JR      LOOP

END:    JR      C,EOF           ;  INDICATES EOF
        (exit routine; otherwise, an error occurred)

CTLBLK: DEFB    1               ;  DRIVE #
        DEFB    -1,-1,-1        ;  BEGINNING OF DIRECTORY
                                   INDEX
        DEFB    -1,-1,-1        ;
        DEFB    Ø               ;  END OF LIST TERMINATOR

MASK:   DEFM    '*/BAS'         ;  FIND /BAS FILES ONLY
        DEFB    ØDH             ;  CR

IMAGE:  DEFS    128             ;  AREA TO STORE DIR RECORD
```

READNX **Function Code 34**
Read Next Record

Reads the record following the current record (the last record accessed). If you just opened the file, READNX reads the first record.

Upon return, your record is in the record area pointed to by RECADR in the parameter list. Or, if the record length is 256 and the record type is fixed, your record is in the area pointed to by BUFADR.

Entry Conditions

A = 34
(DE) = data control block for the currently open file
HL = Reserved for use in later versions of TRSDOS-II

Exit Conditions

NZ = Error
A = error code

RENAME **Function Code 47**

Rename a File

Changes the name and/or extension of a file. You cannot change the password.

Both filespecs (old and new) must be terminated by a carriage return.

If the old filespec is password protected, you must include the password. Do not include a password in the new filespec; TRSDOS-II retains the old password.

Entry Conditions

A = 47
(HL) = old filespec
(DE) = new filespec

Exit Conditions

NZ = Error
A = error code

RETCMD **Function Code 38**
Return after Command

Sends TRSDOS-II an operator command. After the command is completed, your program regains control.

Take care that TRSDOS-II doesn't overlay your program when it loads the command file you choose. Most TRSDOS-II library commands use memory below X'27FF'. A few go up to, but do not include, X'2FFF'. Single-drive copies use all user memory.

Entry Conditions

(HL)	=	command string
B	=	length of string
A	=	38

Exit Conditions

NZ	=	Error
A	=	error code

REWIND **Function Code 48**
Rewind Disk File

Rewinds a disk file. After this SVC is executed, the next read/write accesses the first record of that file.

Entry Conditions
A = 48
DE = data control block for the currently open file

Exit Conditions
NZ = Error
A = error code

RS232C **Function Code 55**

Initialize RS-232C Channel

Sets up or disables either Serial Channel A or B. Before you use it, be sure the correct channel is connected to the modem or other requirements.

This routine sets the standard RS-232C parameters and defines a set of supervisor calls for I/O to the channel you choose. When you initialize Channel A, TRSDOS-II defines the following SVCs: ARCV, ATX, and ACTRL. When you initialize Channel B, TRSDOS-II defines BCRV, BTX, and BCTRL.

Before re-initializing a channel, **always** turn it off. You may **not** turn off Channel A when HOST is active.

Entry Conditions

A = 55
(HL) = parameter list
B = function switch
B = Ø: turn off channel
B ≠ Ø: turn on channnel

Exit Conditions

NZ = Error
A = error code

Parameter List

This 6-byte list includes the required RS-232C parameters:

Channel	Baud Rate	Word Length	Parity	Stop Bits	End-of-List Marker

Channel -- is an ASCII "A" for Channel A or "B" for Channel B.

Baud Rate --is a binary value from "1" to" 8":

1 for 11Ø baud
2 for 15Ø baud
3 for 3ØØ baud

4 for 6ØØ baud
5 for 12ØØ baud
6 for 24ØØ baud
7 for 48ØØ baud
8 for 96ØØ baud

Word Length -- is a binary value from "5" to "8":

5 for 5-bit words
6 for 6-bit words
7 for 7-bit words
8 for 8-bit words

Parity -- is an ASCII "E," "O," or "N" (for "even," "odd," or "none").

Stop bits -- is a binary "1" or "2" for the number of stop bits.

End list marker -- is X'ØØ' for the 16-character default buffer or X'Ø1' for a larger buffer.

If you specify a larger buffer, the next five bytes specify the receive buffer, as shown below:

Byte Ø	End-of-list marker (X'Ø1')
Bytes 1-2	Buffer start address (LSB/MSB)
Bytes 3-4	Buffer end address (LSB/MSB)
Byte 5	Terminator (X'ØØ')

The buffer must be at least 49 bytes long, and the entire buffer (both front and back) must reside below X'8ØØØ'.

To determine the size of the buffer, use the following formula:

$$\frac{\text{number of characters}}{} * 2 + 15 = \text{buffer size}$$

The number of characters to be stored must be at least 17. Remember, it must be small enough to fit below X'8ØØØ'.

SCROLL **Function Code 27**
Protect from Scrolling

Lets you protect part of the display from scrolling. From Ø to 22 lines at the top of the display can be protected. When scrolling occurs, only lines below the protected area are changed.

Entry Conditions

A = 27
B = number of lines to protect (in range [Ø,22])

Exit Conditions

NZ = Error
A = error code

SETBRK **Function Code 3**

Enable and Disable <BREAK> Key

Lets you enable the <BREAK> key by defining a <BREAK> key processing program. Whenever you press the <BREAK> key, your processing program takes over.

On entry to the <BREAK> processing program, the return address of the interrupted routine is on top of the stack. You can return to it with a Z-8Ø return instruction (RET).

All Z-8Ø register contents are unchanged upon entry to the <BREAK> program.

SETBRK also lets you disable the <BREAK> key, by removing the address of the processing program. You cannot change processing programs while the <BREAK> key is enabled.

You must disable the <BREAK> function (HL=Ø) before setting HL to a new address.

The <BREAK> key processing program must reside above X'27FF'.

See "Handling Programmed Interrupts" for programming information.

Entry Conditions

A = 3
HL = address of the <BREAK> key processing program
HL = Ø: disable the <BREAK> function

Exit Conditions

NZ = Error
A = error code

If HL = Ø, then:
(HL) = address of the deleted <BREAK> key processing program

SETUSR **Function Code 2**
Set User

Sets or removes a user vector, thus giving you the ability to add SVC functions. Function codes 96-1Ø1 are available for you to define, unless the serial interface is on (see the RS232C SVC). Function codes 1Ø2-127 are always available for you to define.

Once you add an SVC function, you can call it via the RST 8 instruction, just as you can the TRSDOS-II's SVC routines.

Your routine must reside above X'27FF', and it should end with a RET instruction.

To change an already defined function, you must first remove the old vector.

You can call a non-recursive SVC (system or user-defined) from within a user-defined SVC. For example, do not call SVC 1Ø7 from within SVC 1Ø7.

Entry Conditions

A = 2
B = function code(95 < code < 128)
C = set/reset code
C = Ø: remove the vector
C ≠ Ø: add the vector

If C = Ø (remove vector):
HL = entry address of your routine

Exit Conditions

If C = Ø (remove vector):
HL = removed vector address

SORT **Function Code 56**
RAM Sort

Sorts a list of entries in RAM. All entries must have the same length, from 1 to 255 bytes. The sort is done according to a user-defined sort-key, which you may locate anywhere in the entry.

Entries with duplicate sort-keys remain in the same relative order.

The routine uses a "bubble-sort" algorithm.

Entry Conditions

A = 56
(IX) = first byte of the first entry
(DE) = first byte of the last entry
B = position key within an entry
(Zero equals the first byte of an entry.)
C = length of each entry (C > Ø)
H = sort switch
HL = Ø: use ascending sort
HL ≠ Ø: use descending sort
L = length of sort-key (L > Ø)

In general, B + L must be less than or equal to C.

Exit Conditions

NZ = Error
A = special error return code (not the standard TRSDOS-II code)

Value in A	Meaning
1	Key end exceeds the last byte of the entry
2	Key start exceeds the last byte of the entry
3	Entry length = Ø (invalid)
4	Key length = Ø (invalid)
5	Address of first entry > address of last entry

SOUND **Function Code 6Ø**
Sound Output

Turns on a single tone (approximately 1 khz) in computers that are capable of generating sound.

Entry Conditions
A = 6Ø
HL = sound duration (in 1/3Øth second)

Exit Conditions
None

The sound turns off automatically after n seconds, where:

n = HL/3Ø

n may vary slightly, depending on how much disk and video access is done while SOUND is on.

STCMP **Function Code 22**
String Comparison

Compares two strings to determine their collating sequence.

Entry Conditions

A	=	22
(DE)	=	string1
(HL)	=	string2
BC	=	number of characters to compare

Exit Conditions

Z flag	=	Strings are identical
NZ	=	Strings are not identical
C flag	=	string 1 (DE) precedes string2 (HL) in the collating sequence
A	=	first non-matching character (in string1)

When the strings are not equal, you can get further information from the prime registers, as follows:

HL'	=	address of the first non-matching character in string2
DE'	=	address of the first non-matching character in string1
BC'	=	number of characters remaining This includes the non-matching character.

STSCAN **Function Code 49**
String Scan

Searches through a specified text buffer for the specified string. This string can consist of any values Ø-255 (it is not strictly alphanumeric). The scan stops on the found string or on the first carriage return encountered, whichever comes first.

Entry Conditions

A	=	49
(HL)	=	text area to search
(DE)	=	compare string
B	=	length of the compare string

Exit Conditions

Z flag	=	String is found
NZ	=	Error; check Register A
		A = Ø: string is not found
		A ≠ Ø: error code
(HL)	=	start position of the matching string in the search string

TIMER **Function Code 25**
Set Timer

Lets you set a "seconds clock" to run at the same time as your program and to interrupt the program when time runs out. For example, you can specify the number of seconds for keyboard input. If input is not made within the time limit, TRSDOS-II interrupts the keyboard input routine. It interrupts, also, if you reset the timer.

After the timer counts to zero and causes an interrupt, it shuts off by itself.

See "Programming with TRSDOS-II" for information on interrupts.

Entry Conditions

A = 25
(HL) = SVC to handle interrupt
BC = number of seconds to count down

If HL and BC both = Ø, the timer turns off. If HL = Ø and BC ≠ Ø, the time count resets to the value in BC, and timing continues.

Exit Conditions

None

VDCHAR **Function Code 8**
Video Character

Outputs a character to the current cursor position. It is a scroll mode routine.

Control codes not listed below are ignored.

Key	Hex Code	Function
<F1>	Ø1	Turns on the blinking cursor.
<F2>	Ø2	Turns off the cursor.
<CTRL> <D> or <F3>	Ø4	Turns on the steady cursor.
<CTRL> <G>	Ø7	Sounds bell tone for approximately .25 second (in computers that generate sound).
<BACKSPACE>	Ø8	Moves the cursor back one position and blanks the character at that position.
<TAB>	Ø9	Advances the cursor to the next tab position. Tab positions are at 8-character intervals -- 8, 16, 24, 32, and so on.
<CTRL> <J>	ØA	Moves the cursor down to the next row, in the same column.
<CTRL> <K>	ØB	Moves the cursor to the beginning of the previous line. The cursor does not move into the scroll-protected area.
<ENTER>	ØD	Moves the cursor down to the beginning of the next line.
<CTRL> <N> or <F7>	ØE	Turns on dual routing.

Key	Hex Code	Function
<CTRL> <O>	ØF	Turns off dual routing.
<CTRL> <T>	14	Homes the cursor to the first non-scroll-protected position.
<CTRL> <W>	17	Erases to the end of the line. The cursor doesn't move.
<CTRL> <X>	18	Erases to the end of the screen. The cursor doesn't move.
<CTRL> <Y>	19	Sets the normal display mode (green/white on black). Remains normal until you reset it.
<CTRL> <Z>	1A	Sets the reverse display mode (black on green/white). Remains reverse until you reset it.
<ESC>	1B	Erases the screen and homes the cursor to Position Ø.
<left arrow>	1C	Moves the cursor back one space.
<right arrow>	1D	Moves the cursor forward one space.
<up arrow>	1E	Sets 8Ø characters per line and clears the display.
<down arrow>	1F	Sets 4Ø characters per line and clears the display.

Entry Conditions

A = 8
B = ASCII code for the character to be output to the display. Character codes **must** be in the range [Ø,127].

Exit Conditions

NZ = Error
A = error code

VDGRAF **Function Code 1Ø**
Video Graphics

Displays a buffer of characters, starting at the row and column you specify. It is a graphics mode routine (the cursor "wraps" the display).

Displayable Characters

This routine lets you display the 32 graphics characters (and their reverse images).

The codes are numbered from Ø through X'1F' and are pictured in the operator's manual. Codes X'2Ø' through X'7F' are displayed as standard ASCII characters.

In addition, several special control codes are available:

Hex Code	Result
F9	Sets the normal (green/white on black) mode. The cursor does not move. The mode returns to its previous state after the VDGRAF call is completed.
FA	Sets the reverse (black on green/white) mode. The cursor does not move. The mode returns to its previous state after the VDGRAF call is completed.
FB	Homes the cursor (Row Ø, Column Ø).
FC	Moves the cursor back one space. Column = Column - 1. When the column = Ø, the cursor wraps to Column 79 on the preceding row.
FD	Moves the cursor forward one space. Column = Column + 1. When the column = 79, cursor the wraps to Column Ø on the next row.
FE	Moves the cursor up one row. Row = Row - 1. When the row = Ø, the cursor wraps to Row 23.
FF	Moves the cursor down one row. Row = Row + 1. When the row = 23, the cursor wraps to Row Ø.

At exit, the cursor always is set automatically to the graphics position immediately after the last character

displayed. If the buffer length = Ø, the cursor is set to the position specified in Register Pair BC.

Entry Conditions

A = 1Ø
B = row on the screen to start displaying the buffer
B < 24. For B > 23, use B modulo 24* as the row position.
C = column on the screen to start displaying the buffer
In the 8Ø characters-per-line mode, C < 8Ø. For C > 79, use C modulo 8Ø as the column position.
In the 4Ø characters-per-line mode, C < 4Ø. For C > 39, use C modulo 4Ø as the column position.
D = buffer length (in range [Ø,255])
(HL) = beginning of the text buffer
The buffer should contain codes below X'8Ø' or the special control codes above X'F8'. Any value outside these ranges causes an error.

Exit Conditions

NZ = Error (invalid character sent)
A = error code

*MODULO - A cyclical counting system, in which number1 modulo number2 is the integral remainder after division of number1 by number2. For example: 35 modulo 24 = 11.

VDINIT **Function Code 7**
Video Initialization

Blanks the screen and resets the cursor to the upper left corner (Position Ø in the scroll mode illustration at the beginning of this section). Call VDINIT once before starting any I/O to the display.

Entry Conditions

A = 7
B = character size switch
  B = Ø: 4Ø characters per line
  B ≠ Ø: 8Ø characters per line
C = normal/reverse switch
  C = Ø: reverse mode
  C ≠ Ø: normal mode

Exit Conditions

NZ = Error
A = error code

VDLINE **Function Code 9**
Video Line

Writes a buffer of data to the display, starting at the current cursor position. The buffer should contain ASCII codes in the range [Ø,127].

Control codes that are greater than than X'2Ø' are ignored. (For more information, see VDCHAR.)

VDLINE is a scroll mode routine.

Entry Conditions

A = 9
(HL) = beginning of the text buffer
B = number of characters to be sent
C = end-of-line character
This character is sent to the display after the buffer text.

Exit Conditions

NZ = Error
A = error code

In case of an error:

B = number of characters **not** displayed, including the one causing the error
C = character causing the error

Upon return, the cursor is set automatically to the position following the last character displayed.

VDREAD **Function Code 11**
Video Read

Reads characters from the display into the buffer you specify. It is a graphics mode routine; when it reads past the last column, it wraps back to Column Ø on the next row. When it reads past Column 79 on Row 23, it wraps back to Column Ø, Row Ø.

TRSDOS-II reads reverse (black on green/white) mode characters in ASCII codes, just as it does their normal counterparts. Reverse mode is indicated when the most significant bit (bit 7) is set.

VDREAD can also be used to locate the cursor (see below).

Entry Conditions

A = 11
B = row on screen where the read starts, B < 24
If B > 23, use B modulo 24 as the row position.
C = column on screen where the read starts
In the 8Ø characters-per-line mode, C < 8Ø. For C > 79, use C modulo 8Ø as the column position.
In the 4Ø characters-per-line mode, C < 4Ø. For C > 39, use C modulo 4Ø as the column position.
D = buffer length (in range [Ø,255])
D = Ø: return the cursor position
D ≠ Ø: read the display (B = row, C = column)
(HL) = address of the text buffer

Exit Conditions

BC = current cursor position (B = row, C = column)
The cursor position at exit is the same as at entry; VDREAD doesn't change it.
NZ = Error
A = error code

VIDKEY **Function Code 12**
Combines VDLINE and KBLINE

Sends a prompting message to the display and then waits for a line from the keyboard. VIDKEY is a scroll mode routine which combines the functions of VDLINE and KBLINE.

Starting at the current cursor position, it displays the text buffer you specify. The buffer must contain codes less than X'8Ø'. (See the VDLINE SVC for a list of received control codes and other details.)

Once the text message is displayed, TRSDOS-II positions the cursor immediately after the last character displayed. (To move it to another position, you must place a control code at the end of the text buffer.)

VIDKEY then uses KBLINE to get a line from the keyboard.

(See the KBLINE SVC for a list of received control codes and other details.)

Entry Conditions

A = 12
B = number of characters to be displayed (in the range [Ø,255])
C = length of the keyboard input field (in the range [Ø,255])
(HL) = beginning of the text buffer containing the display message
(DE) = beginning of the text buffer where the keyboard input is to be stored

Exit Conditions

NZ = Error
A = error code

If Z is set (no error):

B = number of characters input from keyboard, including carriage return, if any
C = keyboard line terminator
C = Ø: input buffer was filled
C ≠ Ø: C contains the control character that ended the line (carriage return)

If Z is not set (error):
B = number of characters not displayed, including the one causing the error
C = character causing the error

VIDRAM **Function Code 94**
Transfer Video Display to RAM or Vice-Versa

Copies a screenful of graphics and/or text from a RAM buffer to the video display, or vice-versa. The programmer must be aware of the current display mode -- 8Ø characters per line or 4Ø characters per line.

Every possible data byte is treated as a displayable character. There are no cursor position or other control codes. The following table summarizes the character/code relationships for this routine:

Hex Code	Display Character
ØØ-1F	Normal Graphics
2Ø-7F	Normal ASCII Text
8Ø-9F	Reversed Graphics
AØ-FF	Reversed ASCII Text

Entry Conditions

A = 94
B = function code
B = Ø: copy from RAM to video
B ≠ Ø: copy from video to RAM
(HL) = RAM buffer The start of the buffer must be above X'28ØØ'; the end must be below X'FØØØ'.
If the video is in the 8Ø characters-per-line mode, the buffer must be 8Ø * 24 = 192Ø bytes long.
If the video is in the 4Ø characters-per-line mode, the buffer must be 4Ø * 24 = 96Ø bytes long.

Exit Conditions

None
(VIDRAM does not change the cursor position.)

WILD **Function Code 51**
Wildcard Parser

Compares a filespec with a wildcard spec, depending upon the contents of B, the function code entry condition.

A mask is a filespec with asterisk(s) inserted in one or more positions in the name or extension. An asterisk means "one or more characters -- don't care what they are." For example:

- */BAS masks all files except those that have the extension /BAS.
- *LST masks all files except those that have a filename ending in LST and no extension.

The filespec is a standard TRSDOS-II filespec, terminated by a carriage return.

For details on wildcard specs, see "Wildcard" in the "Introduction."

Contents of B	Result
Ø	Sets the wildcard mask
1	Compares the filespec with the mask
2	Combines a separate name and extension into a filespec

Note: Calling any other TRSDOS-II SVC may clear the mask. Therefore, you should use this procedure:

1. Get the filespec(s) to be checked.
2. Set the mask by calling WILD with B = Ø.
3. Compare the filespec with the mask by calling WILD with B = 1.
4. Repeat Step 3 until all filespecs have been checked.
5. Each time you call another TRSDOS-II SVC, you may have to reset the mask.

Entry Conditions

A = 51
B = function switch
B = Ø: set the wildcard mask
B = 1: compare the filespec to the mask
B = 2: combine into filespec

If B = Ø:
(HL) = wildcard mask

If B = 1:
(HL) = filespec

If B = 2:
(HL) = 11-byte buffer
Bytes Ø-7 = filename; bytes 8-1Ø = extension
You must justify both fields on the left, leaving trailing spaces as needed to fill the field. The format is:

N	A	M	E	Ø	Ø	Ø	Ø	E	X	T

The symbol Ø represents a blank space.

(DE) = 13-byte destination buffer
This buffer is to hold the compressed filespec contained in (HL).

Exit Conditions

If B = Ø:
NZ = Invalid mask specification

If B = 1:
NZ = File does not match or you have set no mask

If B = 2:
(DE) = 13-byte buffer containing filespec
The filespec is created from the 11-byte source text.

WRITNX **Function Code 43**
Write Next Record

Writes the record following the last record accessed. If WRITNX is the first access after the file is opened, TRSDOS-II writes the first record.

Before calling WRITNX, put your record in the record area pointed to by RECADR in the parameter list. Or, if the record length is 256 and the record type is fixed, your record is in the area pointed to by BUFADR.

Entry Conditions

A = 43
(DE) = data control block for the currently open file
HL = Reserved for use in later versions of TRSDOS-II

Exit Conditions

NZ = Error
A = error code